**UNIVERSIDAD AUTÓNOMA DE MADRID**

**ESCUELA POLITÉCNICA SUPERIOR**




**Grado en Ingeniería Informática**

# TRABAJO FIN DE GRADO

## Aplicación web para ayuda en el aprendizaje de la gestión de memoria dinámica en programación con el lenguaje C


**Carlos Mesón de Arana**
**Tutor: Marina De La Cruz Echeandía**
**Ponente: Alfonso Ortega de La Puente**


**JUNIO 2017**

# Aplicación web para ayuda en el aprendizaje de la gestión de memoria dinámica en programación con el lenguaje C

**AUTOR: Carlos Mesón de Arana**
**TUTOR: Marina De La Cruz Echeandía**

**Dpto. Ingeniería Informática**
**Escuela Politécnica Superior**
**Universidad Autónoma de Madrid**
**Junio 2017**

# Resumen

Este Trabajo Fin de Grado surge con objetivo de garantizar una herramienta que muestre visualmente mediante un formalismo de alto nivel distinto del lenguaje de programación C las peculiaridades de la gestión de la memoria dinámica para facilitar el aprendizaje de los estudiantes.

Este proyecto nace de la reiteración de la experiencia observada en las aulas y los laboratorios del incremento de la competencia en la correcta gestión de la memoria dinámica de aquellos alumnos que son capaces de imaginar visualmente el proceso.

El proyecto consistirá en el desarrollo de una aplicación web para facilitar la adquisición de esa imagen visual. Para ello se incorporará al sitio web un canvas en el que diseñar visualmente su algoritmo de gestión de memoria mediante bloques propios desarrollados con el lenguaje de bloques de Google Blockly, un canvas en el que se representará gráficamente la memoria del sistema y las modificaciones que las operaciones de gestión de memoria realicen sobre ella al ser ejecutadas y un área de texto en la que se mostrará la equivalencia en lenguaje C de las operaciones diseñadas de manera visual.

# Abstract

This degree's final project started as a initiative to provide a visual programming tool for learning the principles of dynamic memory management.

This project tries to develop a web application to ease the adquisition of dynamic memory management visualization. For that reason, this website will include a canvas for designing an algorithm of memory management created by Blockly blocks, and another canvas which will have the visual representation of this algorithm using memory blocks. Finally, it will have a textarea representing the C code of that blocks.

# Palabras clave

AngularJS, Blockly, Programación visual, C, memoria dinámica, innovación docente.

# Keywords

AngularJS, Blockly, Visual programming, C, dynamic memory, educational innovation.

*Agradecimientos*

***En primer lugar agradecer a mis tutores Marina de la Cruz y Alfonso Ortega por darme la oportunidad de realizar este Trabajo de Fin de Grado.***

***Por último y más importante, a mi familia que me han apoyado en todo momento y en concreto a mi madre que sin tener conocimientos de informática, me ha ayudado en todo lo posible.***

***Carlos Valerio Mesón de Arana.***

***Junio 2017***

# ÍNDICE DE CONTENIDOS

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# 1 Introducción

## 1.1 Motivación

Tras la experiencia docente del tutor y ponente de este trabajo de varios años, hemos observado que tal vez los alumnos agradezcan una herramienta que muestre visualmente mediante un formalismo de alto nivel distinto del lenguaje de programación C las peculiaridades de la gestión de la memoria dinámica. Este proyecto nace de la experiencia observada en las aulas y los laboratorios del incremento de la competencia en la correcta gestión de la memoria dinámica de aquellos alumnos que son capaces de imaginar visualmente el proceso.

Tal y como se expuso en el resumen del proyecto consistirá en el desarrollo de una aplicación web que intentará facilitar la adquisición de esa imagen visual. Para ello se incorporará al sitio web un canvas en el que diseñar visualmente su algoritmo de gestión de memoria mediante bloques propios desarrollados con el lenguaje de bloques de Google Blockly, un canvas en el que se representará gráficamente la memoria del sistema y las modificaciones que las operaciones de gestión de memoria realicen sobre ella al ser ejecutadas y un área de texto en la que se mostrará la equivalencia en lenguaje C de las operaciones diseñadas de manera visual.

## 1.2 Objetivos

El principal objetivo de este TFG es definir una aplicación web que constituya una herramienta eficiente para el aprendizaje y visualización de la gestión de memoria dinámica en C.

Queremos que sea capaz de definir de la forma más clara posible los conceptos de punteros (diferenciando dirección y valor) así comos los de estructuras. Para ello, haremos uso de los conceptos relacionados con los lenguajes de programación visual, siendo capaces de generar un algoritmo mediante elementos gráficos (en nuestro caso utilizaremos un lenguaje de bloques) que sea transformable en sus equivalentes en código C y en elementos gráficos que representen el estado de la memoria del sistema.

En la propuesta de trabajo inicial que ha demostrado ser demasiado ambiciosa para solo un TFG, se quería que el sitio web fuera capaz de gestionar los algoritmos asociados con una aplicación completa C como las que los alumnos tienen que desarrollar en las asignaturas en las que aprenden tipos abstractos de datos (programación 2 en esta titulación). Sin embargo, al evaluar el esfuerzo necesario para ello se tomó enseguida la decisión de que esta primera versión se ciñera solamente a la gestión de las estructuras de datos involucradas y posponer a trabajos futuros la parte relacionada con las funciones para manipularlas.

## 1.3 Organización de la memoria

Esta memoria está estructurada en secciones que describen todas las fases necesarias para entender el desarrollo de este TFG dentro del proyecto general. Cada sección está compuesta a su vez de subsecciones en las que se especifican con detalle los temas que se están tratando. Las secciones principales están compuestas por los siguientes puntos:

- **Estado del arte:** Se comenta sobre las actuales plataformas web de aprendizaje así como de las nuevas tecnologías de programación web que se han usado y las mejoras que proporcionarán.

- **Análisis:** En esta sección se hace un repaso del conjunto de módulos en los que se define la aplicación así como una comparativa del conjunto de tecnologías disponibles que podíamos usar y la razón por la cual elegimos unas frente a otras.

- **Diseño:** En esta sección se explica el diseño y estructura de módulos que se han llevado a cabo. También se comenta cómo está organizada toda la aplicación y cuales han sido los bloques desarrollados en este TFG sobre el proyecto. Y por último la descripción de interfaces de usuario con algunas maquetas.

- **Desarrollo:** Se explica cómo se ha procedido para el desarrollo de los módulos y las investigaciones que han sido necesarias para poder realizarlos.

- **Pruebas y Resultados:** Conjunto de pruebas para garantizar el correcto funcionamiento de los distintos módulos y la comunicación entre éstos.

- **Conclusiones y Trabajo Futuro:** Balance total del trabajo realizado y el futuro de la aplicación en un ambiente real.

# 2 Estado del arte

## 2.1 Introducción

En el contexto del entorno web, tenemos infinidad de recursos y entornos que nos proveen de herramientas de fácil acceso para aprender nuevos lenguajes de programación. A todos se nos viene a la cabeza sitios de ámbito general (varios lenguajes de programación) como **w3schools.com [10], code_academy[11]** o **code school[12]** y otros de ámbito particular (un único lenguaje de programación) como el tutorial de **AngularJS.**

A pesar de que a medida que pasan los días es mayor la variedad de este tipo de herramientas, su aparición está estrechamente ligada con la demanda, es decir, con la necesidad e importancia que tengan. Es por eso que cuando necesitamos buscar

información relevante sobre lenguajes como **Javascript** o **C#**, encontramos infinidad de recursos, incluyendo recursos gráficos como vídeos. Sin embargo, para lenguajes como **C**, la variedad de información es mucho más escasa y carecemos de herramientas de aprendizaje adecuadas y éstas suelen ser muy básicas (en este caso por ejemplo **cplusplus[13]**).

En la enseñanza, C constituye un aprendizaje básico de los alumnos, y la base para desarrollar conceptos como el de memoria dinámica, es por ello que este TFG está vinculado con un proyecto de innovación docente de la escuela "Metodologías activas y colaborativas y su evaluación en disciplinas científico técnicas: programación de ordenadores" de 2016-17.

## 2.2 Tecnologías y lenguajes usados

En esta subsección se lleva a cabo un análisis y explicación de las tecnologías que se han utilizado para posteriormente entender mejor las secciones de diseño y desarrollo. Principalmente se comentan los lenguajes modernos de desarrollo web que se han usado como principal punto de atracción, sus ventajas y un repaso general de su funcionamiento.

### 2.2.1 AngularJS [1]

Es un framework JavaScript para el desarrollo de aplicaciones web front-end creado por Google. Nació en 2009 pero empezó a ser popular a finales de 2012. Se basa en la filosofía Single Page Application (SPA) que utiliza una única página web cargada completamente sin necesidad de refrescarla, lo que resulta en una web más ligera y con menos llamadas al Servidor. Anteriormente se usaba JQuery para la edición de la vista con funciones JavaScript que se añadían según la necesidad sin seguir ningún patrón, esto resultaba difícil de entender y mantener debido al desorden de código que desembocaba en el temido *SpaguettiCode*.

AngularJS nos ofrece un modelo de organización mucho más efectivo y fácil para desarrollar el código basado en el patrón MVC (Model-View-Controller) aunque ellos se definen como MVW (Model-View-Whatever Works for you). A esto se le añade la sincronización bidireccional (Two-way Data Binding) que permite sincronizar la vista y el modelo para que se puedan modificar los datos desde ambos sentidos.

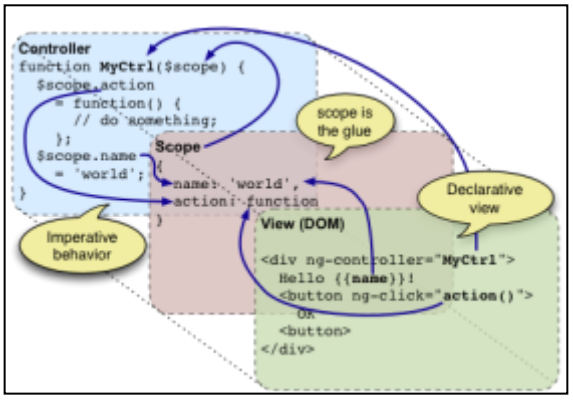



**Figura 2-1: Estructura de modelo/vista/controlador AngularJS**

¿Cómo es posible este mecanismo?, extendiendo la sintaxis HTML de la vista mediante las nuevas directivas proporcionadas por el framework. ***AngularJS*** contiene decenas de directivas que nos proporcionan infinidad de ventajas, veamos un sencillo ejemplo usando la directiva ***ng-model*** para entender el patrón MVC *Figura 2-1*.

### 2.2.2 Lenguajes de programación visual. Blockly[2]

Los lenguajes de programación visual[3] permiten a los usuarios crear programas manipulando elementos gráficamente en lugar de especificarlos textualmente.

Actualmente este tipo de lenguajes son muy importantes en el ámbito de la enseñanza, no sólo para la gente adulta sino también para los niños y adolescentes, pues es un tipo de programación más intuitiva y accesible. Los principales lenguajes que nos encontramos son tres: Scratch[21] , Snap![22] y Blockly[4]. Los dos primeros ofrecen un sitio web en el que poder desarrollar proyectos, mientras que Blockly de Google surge con la pretensión de que cualquiera pueda añadir fácilmente a su sitio web una sintaxis basada en bloques.

*Scratch* es del MIT de sus proyectos de enseñanza de programación sobre todo a niños. *Snap!* es de la universidad de Berckley con la intención de poder introducir en cualquier aspecto de la programación a cualquier persona ajena a ella independientemente de su nivel y su formación. Es un proyecto más experimental que *Scratch* en constante desarrollo y que incluye aspectos de orientación a objetos y programación funcional. Por su parte *Blockly* es la propuesta de Google y tiene una función más instrumental como parte de las extensiones a las web que ofrece. Está pensada como una caja de herramientas "inyectable" en cualquier web que quiera especificar como entrada un algoritmo con un lenguaje de programación de bloques.

La elección de este último en nuestro proyecto ha sido porque sólo necesitábamos implementar la sintaxis de los lenguajes en bloques que fuera abierto y extensible, no queríamos un entorno de desarrollo completo.

Blockly es una librería que añade un editor de código visual web y para aplicaciones Android. Este editor utiliza bloques que se interconectan para representar conceptos como variables, expresiones lógicas, bucles y más.

Estructuralmente hablando, Blockly se compone de un svg que se denomina workspace, y este a su vez se divide en dos partes:

**Toolbox:** Define el conjunto de bloques organizados por categorías que se podrán conectar entre sí.

**Canvas:** Área del svg donde se pueden insertar los bloques. Los bloques que no se encuentren en el canvas no se tendrán en cuenta a la hora de generar el código.

### 2.2.3 Bootstrap [5]

Es un framework CSS desarrollado por Twitter para el diseño de aplicaciones web que permite aplicar estilos mejorando la experiencia del usuario, contiene librerías CSS que incluyen plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos basados en HTML y CSS. Ofrece un mecanismo GRID basado en columnas que permiten realizar páginas web totalmente "responsive" multiplataforma, es decir, se adaptan al dispositivo y pantalla que las ejecute mostrando interfaces de usuario limpias. Además es compatible con la mayoría de navegadores (incluido IE). La forma de usarlo es fijar las clases a elementos de la vista HTML así como tags o directivas para determinadas apariencias de diseño deseadas.

### 2.2.4 RafaelJS [6]

Cuando queremos representar datos en un svg, podemos utilizar las funciones básicas que provee html y javascript, pero es una estructura muy básica que hace casi imposible crear elementos gráficos complejos e interactivos. Por ello, necesitamos del uso de librerías gráficas externas, encontrándonos con algunas como ***d3[14]***, ***GoJS[15]***, ***SVG.js[16]***…

En nuestro proyecto queríamos una librería que aportase la funcionalidad básica para poder gestionar los conceptos de bloques de memoria de la forma más sencilla posible, y por eso nos decidimos finalmente por ***RafaelJS[6]***.

Raphael.js es una librería Javascript que permite dibujar gráficos vectoriales para páginas web utilizando SVG, y trata a los elementos del canvas como si fuesen objetos.

### 2.2.5 CKEditor [7]

En lo referente a los editores de texto, podemos encontrarnos con cosas tan simples como un textarea contenido en una página html como con un editor de texto rico y complejo como ***CKEditor[7]*** o ***TinyMCE[17]***.

Para nuestro proyecto necesitábamos de uno que permitiese mostrar el texto con resaltado de código, y es por ello que decidimos utilizar ***CKEditor*** pues era el editor con el que más habíamos trabajado y con el que teníamos más experiencia.

## *2.3 Conclusiones*

Aun habiendo muchas posibles librerías y tecnologías entre las que poder elegir para poder desarrollar este TFG, al final optamos por utilizar aquellas con el mejor equilibrio entre utilidad y sencillez, pues era importante garantizar que éstas fueran capaces de comunicarse entre sí, pues, cuanto más compleja o completa, más difícil habría sido extraer los datos para poder interpretarlo con las demás.

# 3 Análisis

En esta sección se expone el análisis que se ha realizado para poder definir las unidades atómicas en forma de unidades de Blockly así como su representación en código c y visualización gráfica garantizando las funciones básicas de gestión de memoria dinámica.

En un principio se ha realizado una fase de investigación preguntando a los distintos profesores las principales dificultades a las que se enfrentaban los alumnos a la hora de comprender el funcionamiento de los punteros y la memoria para la gestión dinámica de datos en C.

A continuación se ha llevado a cabo una segunda fase en la que se ha preguntado a distintos alumnos cuál de las distintas formas de representación gráfica de la memoria es la que les parecía más intuitiva.

Todas las fases de este análisis se han realizado desde un punto de vista informal y superficial, no representando el posible coste de esfuerzo/tiempo real del proyecto de innovación docente.

## *3.1 Descripción del proyecto*

### 3.1.1 Subsistemas

#### 3.1.1.1 Blockly

En lo referente al bloque de representación, nos encontramos con la posibilidad de utilizar ***Scratch***, ***Snap!*** o ***Blockly***. Las dos primeras opciones constituye una librería mucho más completa, pues incluye un **editor visual, intérprete** y una **interfaz de usuario**, mientras que Blockly únicamente provee de las herramientas necesarias para generar la primera opción.

##### 3.1.1.1.1 Comparativa

| Scratch | |
|---|---|
| **PROS** | **CONTRAS** |
| • Programa gratuito y de uso libre.<br>• Disponible en varios sistemas operativos.<br>• Permite compartir los proyectos a través de la web.<br>• Es multilenguaje. | • Utiliza Flash, lo que provoca problemas de visualización y de integración.<br>• No es extensible, de forma que no se puede comunicar con otras librerías. |

**Tabla 3-1: Pros/Contras lenguaje Scratch**

| Snap | |
|---|---|
| **PROS** | **CONTRAS** |
| • Programa gratuito y de uso libre.<br>• No utiliza Flash (HTML5). | • Sistema de archivos/datos complejo, difícil de hacer extensible con otras librerías.<br>• Poca presencia online. Poca documentación.<br>• Editor gráfico integrado muy sencillo. |

**Tabla 3-2: Pros/Contras lenguaje Snap**

| Blockly | |
|---|---|
| **PROS** | **CONTRAS** |
| • Programa gratuito y de uso libre.<br>• No utiliza Flash (HTML5).<br>• Extensible, código abierto.<br>• Fácil de insertar en tu propia web. | • Únicamente define la interfaz de generación de bloques.<br>• El intérprete de Blockly no permite definir tipos a las variables.<br>• No es un lenguaje propiamente dicho, sino más bien una API.<br>• Documentación prácticamente inexistente, dificulta el desarrollo. |

**Tabla 3-3: Pros/Contras lenguaje Blockly**

La comparativa de las tres plataformas puede resumirse de la siguiente manera. Desde el punto de vista de la funcionalidad *Scratch* y *Snap!* proporcionan un entorno de desarrollo completo de aplicaciones concebidas como agentes virtuales gráficos que interactúan entre ellos y con el usuario en un canvas gráfico que es lo que se muestra cuando el programa se ejecuta. Cada agente tiene su código (codificado con bloques) y la posibilidad de generar y atender eventos de interacción con el canvas gráfico, con el resto de agentes y con el usuario. Además cada agente tiene un aspecto gráfico modificable. El paradigma de diseño de la aplicación es concurrente respecto a la generación y atención de los eventos que son los que rigen el flujo de programa de la aplicación. La plataforma de desarrollo ofrece acceso a los bloques disponibles para especificar los algoritmos así como múltiples facilidades que incrementen la ergonomía del proceso de programación. Desde otra perspectiva *Snap!* y *Blockly* son de código abierto.

Por todo esto, finalmente decidimos optar por utilizar ***Blockly***, que proporciona la funcionalidad básica de un lenguaje de programación visual permitiéndonos entrelazarlo con otras librerías.

### 3.1.1.2 CKEditor

En lo que se refiere al subsistema del editor de texto, las dos opciones que teníamos en mente fueron CKEditor y TinyMCE. Este módulo no es de los más importantes, pues solo lo utilizaremos para visualizar el algoritmo definido en Blockly en su equivalente en lenguaje C[8], y finalmente nos decantamos por CKEditor por tener implementado de serie el **resaltado de código** para C (figura 3-1).

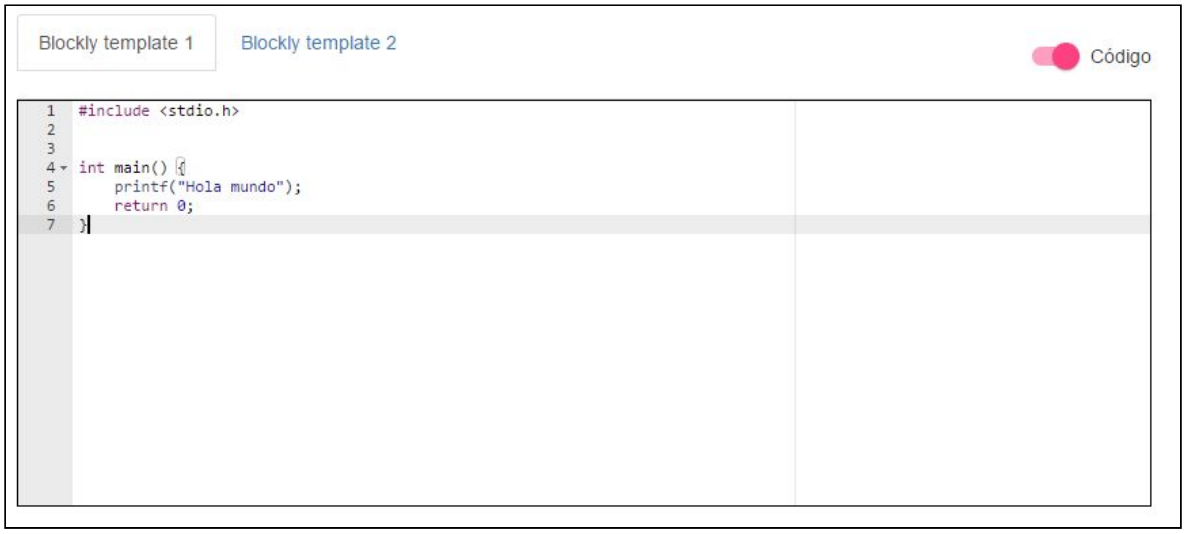



**Figura 3-1: Muestra de CKEditor**

### 3.1.1.3 RafaelJS

Por último, para el subsistema gráfico, vimos varias opciones como oCanvas[18], Three.js[19] y paperjs[20], pero las dos que más nos llamaron la atención fueron d3 y RafaelJS.

| d3 | |
|---|---|
| **PROS** | **CONTRAS** |
| • Muchos ejemplos e información sobre esta librería.<br>• Gran cantidad de funciones y personalización.<br>• Fácil de incrustar en una página web, no depende de otras librerías. | • La generación de gráficos se tiene que definir en base a un sistema de referencia (ejes), lo que dificulta la creación de gráficos que no son estadísticos.<br>• La documentación varía mucho en función de si es d3 versión 3 o d3 versión 4, lo que dificulta la accesibilidad.<br>• Los elementos gráficos se tienen que vincular a datos del modelo. |

**Tabla 3-4: Pros/Contras librería gráfica d3**

| RafaelJS | |
|---|---|
| **PROS** | **CONTRAS** |
| • Sencillo, fácil de aprender.<br>• Definición de elementos gráficos como si se tratase de un lenguaje orientado a objetos, fácil de implementar.<br>• Los elementos gráficos no tienen que estar vinculados a datos del modelo. | • Faltan algunas funciones que dificultan algunas tareas, como el desplazamiento/zoom de los elementos.<br>• Falta de documentación y ejemplos, no es muy popular. |

**Tabla 3-5: Pros/Contras librería gráfica RafaelJS**

A primera vista puede parecer que d3 es una mejor opción para utilizar en nuestro proyecto, pero finalmente nos decantamos por ***RafaelJS*** por su sencillez, ya que en nuestro caso, solo queremos definir elementos gráficos sencillos compuestos por rectángulos, texto y flechas, y la funcionalidad extra que otorga ***d3*** no supone un beneficio tan grande como para que compense la dificultad extra de crear un sistema de referencia y un modelo para que funcione en esta librería.

## *3.2 Requisitos Funcionales*

Por un lado tenemos los requisitos funcionales (RF) separados en tres subsistemas (Módulo Blockly, Módulo Gráfico y Módulo Editor) y el módulo principal, y por otro los requisitos no funcionales (RNF) que se agrupan en común.

### 3.2.1 Módulo Blockly

Este módulo de la aplicación está destinado a proveer de los elementos gráficos en forma de piezas de puzzle que contienen las unidades atómicas básicas del lenguaje C, necesarias para poder generar algoritmos en *Blockly*. Se podrán definir variables, punteros y estructuras así como generar bucles y sentencias condicionales.

[CREARPUNTEROS_RFB_1] **Poder definir punteros y su tipo:** Se definirán los bloques necesarios para poder crear punteros asignándoles un identificador así como un tipo STRING, ENTERO o ESTRUCTURA.

- **Entradas:** Bloques *Blockly* correspondientes a la definición de punteros.
- **Salidas:** Se actualizará el código equivalente C del editor de texto y el canvas gráfico con la representación de los bloques de memoria.
- **Archivos internos:** Información que el sistema guarda internamente para llevar cuenta de los punteros definidos.

[APUNTARPUNTEROS_RFB_2] **Asignar valor a puntero:** Se definirán los bloques necesarios para modificar el valor del puntero para asignarle la dirección correspondiente de una variable, estructura u otro puntero.

- **Entradas:** Bloques *Blockly* correspondientes a la asignación de punteros.

- **Salidas:** Se actualizará el código equivalente C del editor de texto y el canvas gráfico con la representación de los bloques de memoria.
- **Archivos internos:** El sistema actualizará el valor del puntero correspondiente al nuevo establecido en el bloque.

[RESERVARPUNTEROS_RFB_3] **Reservar espacio para puntero:** Se definirán los bloques necesarios para poder reservar espacio en memoria mediante el uso de la función calloc.

- **Entradas:** Bloques *Blockly* correspondientes a la reserva de memoria.
- **Salidas:** Se actualizará el código equivalente C del editor de texto y el canvas gráfico con la representación de los bloques de memoria.
- **Archivos internos:** El sistema actualizará el valor del puntero correspondiente al nuevo establecido en el bloque.

[EXTENSIBILIDAD_RFB_4] **Posibilidad de añadir bloques dinámicamente:** Se definirán las medidas necesarias para permitir añadir bloques al proyecto de forma externa.

- **Entradas:** Archivos javascript y xml necesarios para definir la nueva funcionalidad del bloque a añadir.
- **Salidas:** Se actualizará el conjunto de bloques y toolbox del proyecto para integrar el nuevo bloque.
- **Archivos internos:** El sistema actualizará el conjunto de bloques *Blockly* incluyendo el nuevo definido externamente.

### 3.2.2 Módulo Editor

Este módulo de la aplicación está destinado a proveer de un editor de texto que permita ver el equivalente de la representación en código C.

- [RESALTADOVISUAL_RFE_1] **Resaltado visual de código C:** La representación equivalente en C deberá de tener el correspondiente resaltado visual para facilitar su interpretación.
  - **Entradas:** Irrelevante.
  - **Salidas:** Se actualizará la visualización del código C equivalente de los bloques para resaltar las partes importantes.
  - **Archivos internos:** Irrelevante.

### 3.2.3 Módulo Gráfico

Este módulo es el responsable del canvas en el que se mostrará la representación visual de las operaciones de gestión de memoria que haga el algoritmo especificado mediante blockly.

- [BLOQUESGRAFICOS_RFG_1] **Representación en forma de bloques:** Para facilitar la comprensión de la memoria dinámica, los bloques de memoria de deben representar no tal cual aparecerían en un disco duro, sino como bloques independientes con su correspondiente dirección y valor.
    - **Entradas:** El usuario creará los bloques Blockly correspondientes para generar el algoritmo deseado y deberá aparecer en el svg RafaelJS.
    - **Salidas:** Se actualizará el svg correspondiente con el pertinente conjunto de bloques de memoria que equivalen al algoritmo definido por el usuario.
    - **Archivos internos:** El sistema creará los archivos pertinentes para controlar las dependencias, valores y posiciones de los bloques de memoria que se deben dibujar, así como las posibles conexiones de punteros (dirección-valor).

- [INTERACTIVIDAD_RFG_2] **Elementos gráficos arrastables y escalables:** Con motivo de la escalabilidad y tamaño del proyecto, tiene que ser posible ajustar la vista de la representación gráfica para poder observar detalles concretos o una visión general de todos los datos.
    - **Entradas:** El usuario hará click sobre los elementos gráficos y los arrastrará o moverá la rueda del ratón para modificar el zoom del canvas.
    - **Salidas:** Los elementos gráficos del svg deben desplazarse o hacer zoom en función de las acciones del usuario.
    - **Archivos internos:** Irrelevante.

### 3.2.4 Módulo General

- [INTEGRACIONBLOCKLY_RFM_1] **Integración con las librerías *Blockly*:** Tal y como se explicará en apartados posteriores de la memoria, tuvimos que crear un intérprete propio que fuese capaz de procesar el algoritmo definido en Blockly considerando tipos de variables (pues el intérprete propio de *Blockly* carece de esa capacidad). Este requisito es el que mayor tasa de esfuerzo supone y el más ha impactado en la toma de decisiones del diseño.
    - **Entradas:** El usuario generará el algoritmo *Blockly* correspondiente combinando un conjunto de bloques.
    - **Salidas:** El intérprete se encargará de asignar el resultado de la transformación del algoritmo en lenguaje C y elementos gráficos RafaelJS.
    - **Archivos internos:** El intérprete del sistema controlará el conjunto de instrucciones definidas por el algoritmo *Blockly* y actualizará las variables y punteros en caso de que sea necesario. Además será el encargado de generar en código equivalente en C que será asignado al módulo del editor de texto.

## *3.3 Requisitos No Funcionales*

Los requisitos no funcionales establecen características del diseño e implementación del sistema.

[RESPONSIVE_RNF1] **Diseño *responsive*:** Se tendrá que diseñar una interfaz capaz de adaptarse a los distintos tamaños de pantalla para poderse visualizar correctamente en ordenador, tablets y móviles.

- **Entradas:** El usuario podrá modificar el tamaño de la ventana o utilizará otro tipo de pantalla (con unas medidas distintas).
- **Salidas:** Los componentes gráficos de la página web deben adaptarse al nuevo tamaño de pantalla, utilizando los pertinentes métodos definidos por CSS y Bootstrap.
- **Archivos internos:** Irrelevante.

[COMPATIBLE_RNF1] **Compatible con todos los navegadores:** Se utilizarán funciones y librerías compatibles con al menos los navegadores de internet explorer, chrome y mozilla.

- **Entradas:** Irrelevante
- **Salidas:** Irrelevante
- **Archivos internos:** Irrelevante

## *3.4 Estimaciones software*

A modo de cuantificar superficialmente el esfuerzo del proyecto, y justificar el alcance de éste, utilizaremos la técnica del **Ánalisis de Puntos de Función,** siguiendo el modelo constructivo de Costos **COCOMO II [9]**.

Esta técnica permite cuantificar el tamaño de un sistema en unidades independientes del lenguaje de programación, las metodologías, plataformas y/o tecnologías utilizadas, denominadas **Puntos de Función**.

Lo primero que hay que hacer, en función de los requisitos funcionales y no funcionales definidos, es calcular los **Puntos de Función no-ajustados** en base a los tipos de entradas, salidas y archivos (ver Anexo A).

| | COMPLEJIDAD | | | | | | |
|---|---|---|---|---|---|---|---|
| | Baja | Media | Alta | Baja | Media | Alta | PF No-ajustados (UFP) |
| **Funciones de DATOS** | Frecuencia | | | Peso | | | |
| Internal Logical Files (ILF) | ***3*** | ***2*** | ***0*** | *7* | *10* | *15* | 48 |
| External Interface Files (EIF) | ***0*** | ***0*** | ***0*** | *5* | *7* | *10* | 0 |
| **Funciones TRANSICIONALES** | | | | | | | |
| External Inputs (EI) | ***1*** | ***0*** | ***0*** | *3* | *4* | *6* | 3 |
| External Outputs (EO) | ***0*** | ***0*** | ***0*** | *4* | *5* | *7* | 4 |
| External Queries (EQ) | ***0*** | ***0*** | ***0*** | *3* | *4* | *6* | 0 |
| | | | | | | **TOTAL** | **51,0** |

**Tabla 3-6: Complejidad Entradas/Salidas**

A continuación, calcularemos los **Puntos de Función Ajustados** aplicando un **Factor de Ajuste** basado en la cuantificación de ciertos coeficientes vinculados con las características deseadas del sistema (ver Anexo B). A cada una de estas características se le asigna un factor de peso (un valor entre 0 y 5) que indica la importancia de las características para el sistema bajo análisis.

- **0:** No presento o sin influencia.
- **1:** Influencia incidental.
- **2:** Influencia moderada.
- **3:** Influencia media.
- **4:** Influencia significativa.
- **5:** Fuerte influencia.

| Factor de Complejidad | FC |
|---|---|
| Sistemas distribuidos | 1 |
| Rendimiento o tiempo de respuesta | 2 |
| Eficiencia del usuario final | 5 |
| Procesamiento interno complejo | 2 |
| El código debe ser reutilizable | 4 |
| Facilidad de instalación | 5 |
| Facilidad de uso | 5 |
| Portabilidad | 5 |
| Facilidad de cambio | 2 |
| Concurrencia | 1 |
| Características especiales de seguridad | 1 |
| Provee acceso directo a terceras partes | 1 |
| Facilidades especiales de entrenamiento a usuarios | 4 |
| Facilidad de cambio | 2 |
| **TOTAL** | **40,0** |

**Tabla 3-7: Factor de Complejidad**

Factor de ajuste: **AF = (FC*0.01) + 0,65 = 1,05**
Puntos de función ajustados: **FP = UFP * AF = 53,55**

## 3.5 Organización temporal

Una vez obtenidos los **Puntos de Función Ajustados**, aplicamos coeficientes que convierten este valor a otros, tales como el esfuerzo y el tiempo.

Asumiendo una tasa de esfuerzo de 1,2PF/persona-día (teniendo en cuenta que *Blockly* y *RafaelJS* con librerías con las que nunca antes se había trabajado y con poca documentación).

| Incrementos | PF | Esfuerzo estimado |
|---|---|---|
| Incremento | *51,0* | *61,2* |
| **TOTAL** | **51,0** | **36,51** |

**Tabla 3-8: Incrementos**

En este caso, el proyecto de desarrolla por una única persona, lo que significa que el esfuerzo estimado corresponde directamente con la cantidad de días de duración del proyecto.

| Tarea | Duración (días) |
|---|---|
| Incremento | *62* |
| **TOTAL (días)** | **35** |

**Tabla 3-9: Duración de incrementos**

## 3.6 Conclusiones

Los cálculos de la estimación de esfuerzo y duración se han hecho considerando los requisitos funcionales y no funcionales básicos, excluyendo algunos como la posibilidad de crear y manejar **TAD's** y definir **funciones** (con sus respectivos ámbitos de variables y representación gráfica), que habrían multiplicado exponencialmente la duración de este proyecto. Particularmente, la principal causa que ha fomentado la toma de esta decisión ha sido el requisito de controlar tipos de variables en *Blockly*.

Tal y como se explicará en el apartado de diseño, *Blockly* tiene un intérprete propio capaz de procesar el algoritmo definido por bloques. Sin embargo, no es posible aplicar los conceptos de tipos de variables en él, de forma que tuvimos que crear uno desde el principio la añadir esta funcionalidad (necesaria a la hora de entender conceptos como punteros y estructuras), lo que implicó un aumento del esfuerzo y tiempo necesarios.

Por consiguiente, toda esta funcionalidad importante a la hora de aprender sobre la gestión de memoria dinámica, se planteará como trabajo futuro para un **nuevo TFG**, quedando disponible para el próximo año.

# 4 Diseño

El diseño de un software es una de las fases más importantes del proyecto, que se realiza en función de las necesidades y comunicación entre los distintos módulos (*Blockly*, *CKEditor* y *RafaelJS*). En esta sección se analizará todo lo referente a la arquitectura, cuál es el esqueleto de la aplicación que se tiene como punto de partida y cómo se ha extendido para poder realizar los módulos asociados a este trabajo.

## *4.1 Arquitectura*

### 4.1.1 Arquitectura global de la aplicación

La aplicación se ha desarrollado en **AngularJS** usando su patrón MVC que incluye las particularidades que se comentan en las primeras secciones de este documento, en el que se explica que el modelo puede ser modificado tanto por la vista como por el controlador.

Contiene toda la lógica y visualización que se ejecuta en el Cliente. Al utilizar AngularJS como framework para nuestra aplicación, nuestro proyecto se definirá por un conjunto de ***controllers-templates*** que se comunicarán entre sí.

- **Templates:** Contiene los ficheros (HTML) con la parte visual (Vista) de la aplicación que representa el navegador. Aunque AngularJS funciona con una única página sin recargarla, se desarrolla cada parte de la aplicación en ficheros HTML independientes que se inyectarán y fusionarán dinámicamente en la página principal. Con esto se muestra al usuario las ventanas que solicita en cada momento de igual manera que ocurre con la navegación tradicional donde se redirigen enlaces. Los Template principales tienen asociado un Controller.

- **Controllers:** Contiene los controladores que manejan la lógica de la aplicación en el lado Cliente, modifican e inicializan los datos y ofrecen funciones y comportamientos para los mismos. Los controladores interactuarán con la vista y viceversa y serán los encargados de comunicarse con los Services. Además podremos inyectar dependencias de otros módulos para la reutilización de código. Al igual que los Template distinguiremos cada controlador según partes diferenciadas de la aplicación.

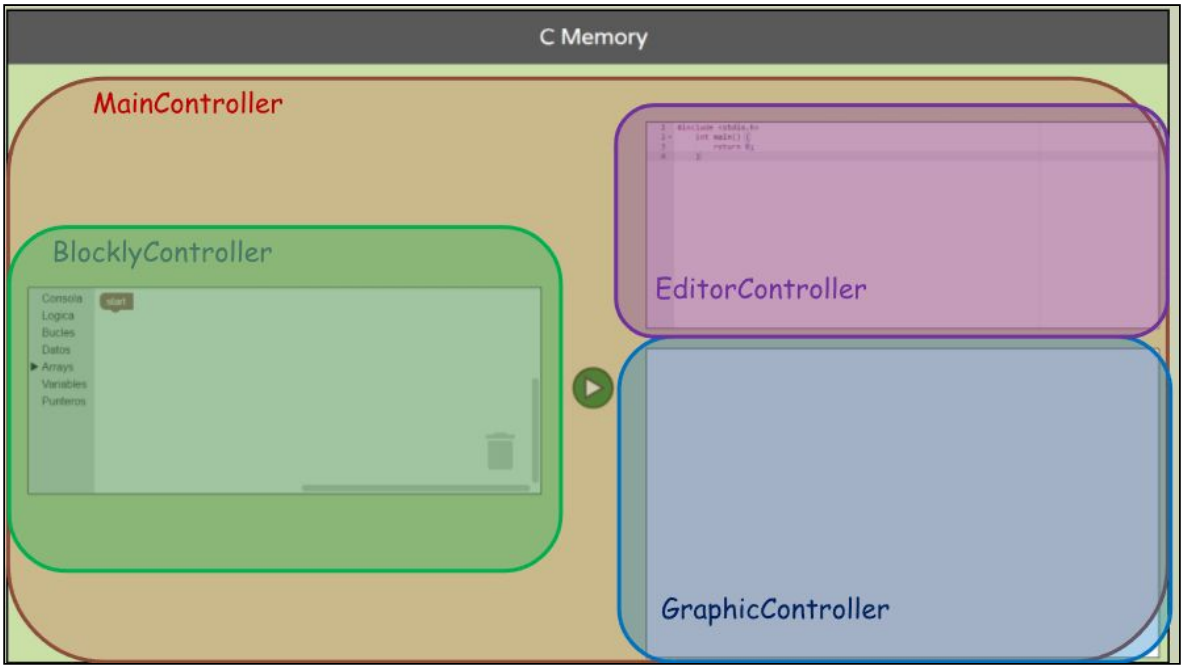



**Figura 4-1: Estructura controladores AngularJS en el proyecto**

Tal y como se muestra en la figura 4-1, en nuestro proyecto tenemos 4 secciones cuyas vistas (templates) quedan definidas por el *** *** html. Asimismo, cada una de estos templates tendrán asociados un controlador que se definirá en el archivo javascript correspondiente.

```
<!-- CONTAINER -->
<div class="generalContainer" ng-app="App" ng-controller="MainController">

    <div class="generalPanel">

        <!-- LEFT -->
        <div class="cell left">
            <div class="editorSwitch">...
            </div>

            <div class="tabsContainer">...
            </div>

            <div ng-show="!showTextEditor" id="blocklyArea" class="blocky" ng-controller="BlocklyController">
            </div>

            <div ng-show="showTextEditor" id="interpreterArea" class="interpreter" ng-controller="EditorController">...
            </div>
        </div>
        <!-- END LEFT -->

        <div class="cell">
            <input type="image" class="btnGenerateCode" src="./rcs/btn_play.png" ng-click="Webc.showCode()" value="">
        </div>

        <!-- RIGHT -->
        <div class="cell right">
            <div class="graphic" ng-controller="GraphicController">
                <div id="c_canvas"></div>
            </div>
        </div>
        <!-- END RIGHT -->

    </div>

    <div ng-show="!showTextEditor" id="blocklyDiv"></div>

</div>
<!-- END CONTAINER -->
```

**Figura 4-2: Vinculación de template-controlador AngularJS**

En este caso, se puede observar en la Figura 4-2, que la división padre tiene asociado el controlador principal ***MainController***, siendo este accesible por los controladores que están contenidos en él. Por consiguiente, toda la funcionalidad común para todos los módulos debe estar contenida en este controlador, mientras que los datos del modelo específico de cada módulo (tales como las funciones para pintar el texto o los bloques de memoria del svg), se definirán en sus respectivos controladores.

### 4.1.2 Modelo de datos

#### 4.1.2.1 Librería gráfica - Blockly

Para este módulo, utilizamos la estructura de Blockly, en la cual se definen para cada bloque una **función de vista** (en la que se definen cómo se pintará el bloque y cómo éste interactuará con otros) y una función de **generación de código** (que define el código que se ejecuta cada vez que aparece una instancia de este bloque en el programa *Blockly*).

Para entender las decisiones de diseño que se han tomado, primero debemos comprender cómo se estructura un proyecto *Blockly* y qué cosas podemos definir para poder modificar el funcionamiento de éste.

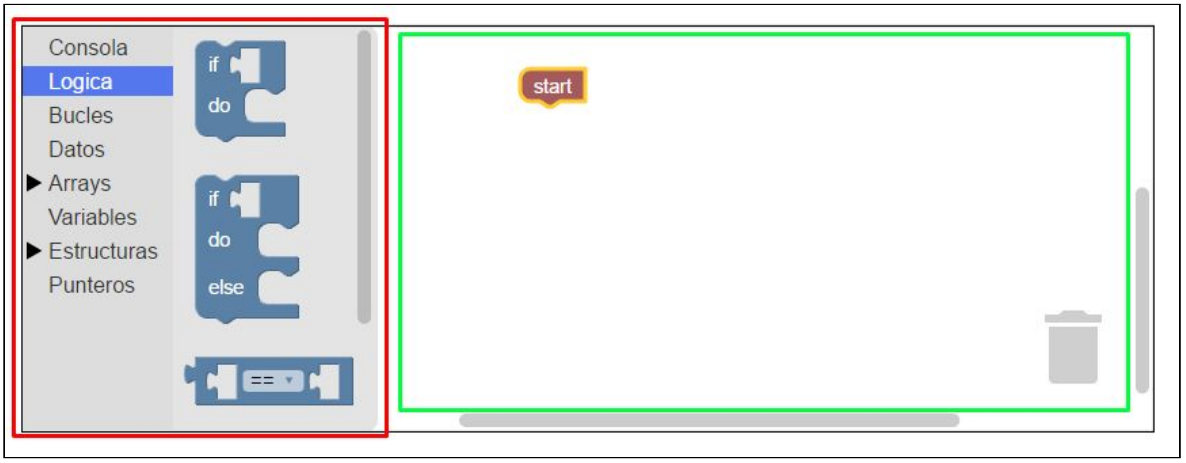



**Figura 4-3: Esquema de proyecto *Blockly***

Tal y como se puede ver en la figura 4-3, un proyecto bloque se divide en 2 partes, el ***toolbox*** y el ***canvas*** donde se insertan los bloques que definen el algoritmo. Un mismo tipo de bloque puede instanciarse todas las veces que se quiera, lo único que hay que hacer es arrastrar el bloque deseado al canvas y formará parte del algoritmo.

Ahora bien, para definir un bloque personalizado, hay que definir antes las 2 partes que se describen en la figura 4-4, pudiendo distinguir 3: La **definición de la vista** (que especifica color, inputs y formato del bloque) y el **generador de código/definición del modelo** (que especifica la funcionalidad del bloque dentro del algoritmo).

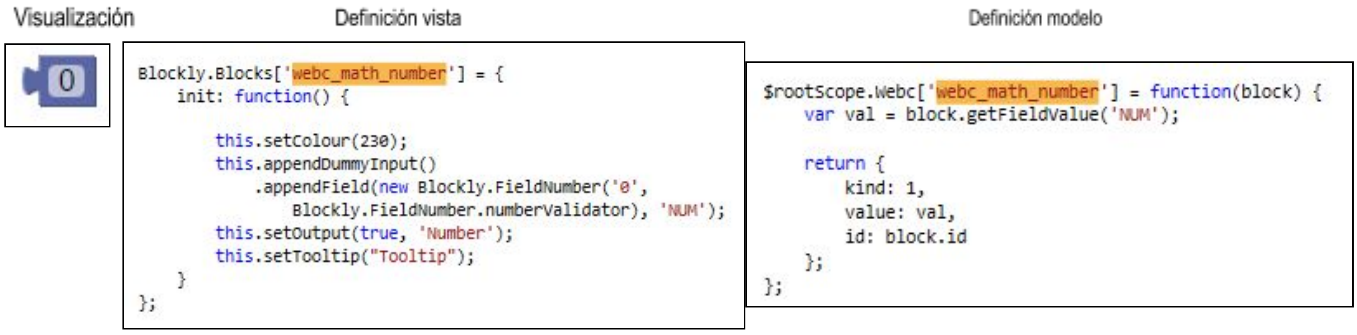



**Figura 4-4: Estructura de definición de bloque *Blockly***

En el caso concreto que se muestra en las imágenes, en la definición de la vista especificamos el color y el tipo de entradas y salidas de este bloque, mientras que en el apartado de definición del modelo (generador de código), declaramos la funcionalidad de este bloque cada vez que aparezca en el algoritmo, que en este caso en concreto, lo único que hace es extraer el valor del campo de la entrada **NUM** y devolver un objeto javascript (que más veremos que se trata de una instrucción).

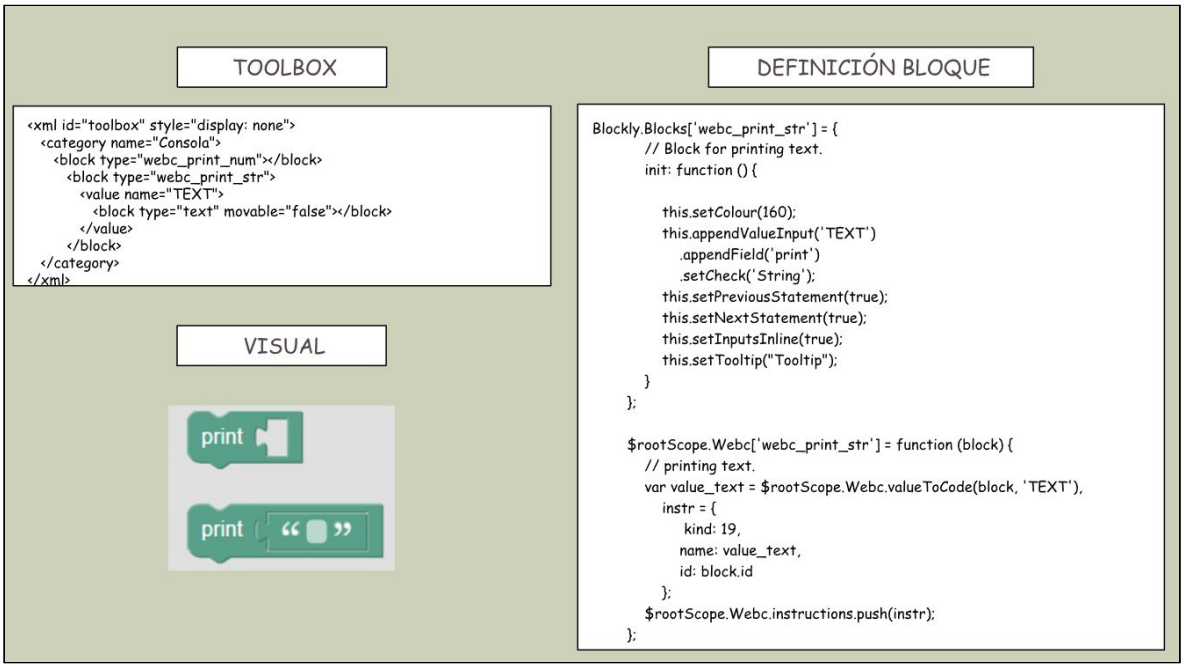



**Figura 4-5: Estructura de definición de bloque *Blockly***

Aun habiendo definido un bloque mediante el conjunto vista/modelo, aún quedaría especificar que queremos que este nuevo bloque aparezca en el **toolbox** de *Blockly* (todos los proyectos *Blockly* tienen un xml definido que especifica el *toolbox*), y eso se hace mediante un xml tal y como el que aparece en la Figura 4-5.

En resumen, para que nuestros bloques personalizados estén disponibles para definir el algoritmo, debemos declarar los 3 componentes básicos: su vista, su modelo y su presencia en el toolbox.

La pregunta es ¿qué se encarga de procesar el algoritmo de *Blockly* y de ejecutar el modelo asociado a cada uno de ellos? Blockly tiene integrado un intérprete propio que se encarga de procesar el algoritmo. De esta forma, si queremos definir bloques que generen código, por ejemplo javascript, podríamos simplemente hacer que los modelos de los bloques devolviesen el equivalente en lenguaje javascript, en el caso de la figura 4-3, **return 'alert(str)'.**

Siendo así, la primera pregunta que viene a la cabeza es por qué entonces nosotros no definimos los modelos de esta forma, devolviendo únicamente el equivalente en **lenguaje C** de cada uno de los bloques. La respuesta es por los tipos de las variables.

En lenguajes que no son **fuertemente tipados**, como *Javascript*, no es necesario controlar el **tipo de las variables**, de forma que su definición es mucho más rápida y sencilla en contraposición a los lenguajes que sí lo son, tales como el **lenguaje C**, que es el que queremos utilizar para definir el algoritmo final de *Blockly*. El problema reside entonces que a la hora de definir tipos de variables, punteros y estructuras necesitamos poder controlar el tipo de estas, cosa que no se puede hacer con el intérprete de *Blockly* (que únicamente gestiona las variables con los identificadores).

Por consiguiente, nos vimos obligados a crear un **intérprete propio**, que únicamente utilizase el de Blockly para procesar los bloques del algoritmo, para nosotros procesarlo después haciendo uso de un **sistema de instrucciones**. Este sistema es semejante al de un procesador con su correspondiente conjunto de instrucciones. Nuestro algoritmo *Blockly* procesará cada uno de los bloques que definirán en su modelo una instrucción (un objeto javascript como el que se devuelve en la figura 4-4, que posee un conjunto de atributos, como los que están definidos en la figura 4-6).

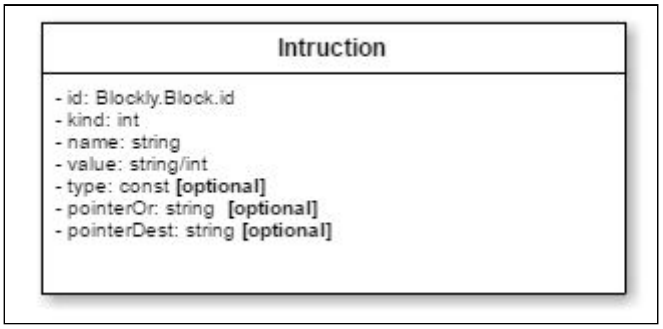



**Figura 4-6: Diagrama de clase módulo Blockly clase instrucción**

Estas instrucciones se almacenarán en un array, que finalmente será recorrido en bucle para, en función de la instrucción recibida, procesarla de una manera u otra, en nuestro caso, generando la correspondiente versión en lenguaje C y en elementos gráficos de bloques de memoria (en *RafaelJS*).

Dependiendo del tipo de instrucción, habrá campos (los que aparecen definidos en la figura 4-6) que sean necesarios o no, por ejemplo, en el caso de una instrucción de tipo impresión (printf), los campos pointerOr y pointerDest no son necesarios, mientras que en un bloque de tipo asignación de puntero sí que lo son.

### 4.1.2.2 Librería gráfica - RafaelJS

Una vez procesado el algoritmo, y generadas las instrucciones pertinentes, debemos interpretar éstas para generar el equivalente gráfico en bloques de memoria en *RafaelJS*, que poseen el formato que aparece en la figura 4-7.


**Figura 4-7: Formato de bloque de memoria**

Un bloque de memoria se compone de un **tag**, que identifica a la variable, puntero o estructura, una **dirección** (ubicación en la memoria) y un **valor** (el contenido que hay en esa ubicación de la memoria). Utilizando esta abstracción, definiremos tantos bloques como variables, punteros y estructuras se hayan definido en el algoritmo *Blockly*. El conjunto de funciones y objetos necesarios

En caso de encontrarnos con una estructura o un array, el tag de la variable referenciaría a más de una dirección, es decir, el contenido del campo **valor** estaría constituído por más de un bloque de memoria (figura 4-8).

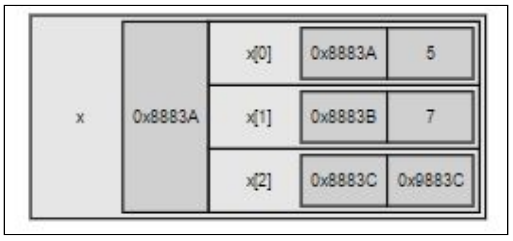



**Figura 4-8: Representación de array mediante bloques de memoria**

Como se puede observar, el valor de la dirección del **bloque padre** coincide con la dirección del primer bloque de su contenido.

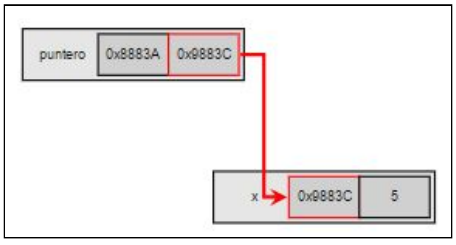



**Figura 4-9: Representación de puntero mediante bloques de memoria**

Por último, para representar punteros, decidimos utilizar flechas indicando la dirección del contenido a la que apunta, pudiendo así crear listas enlazadas (tal y como se muestra en la imágen 4-9).

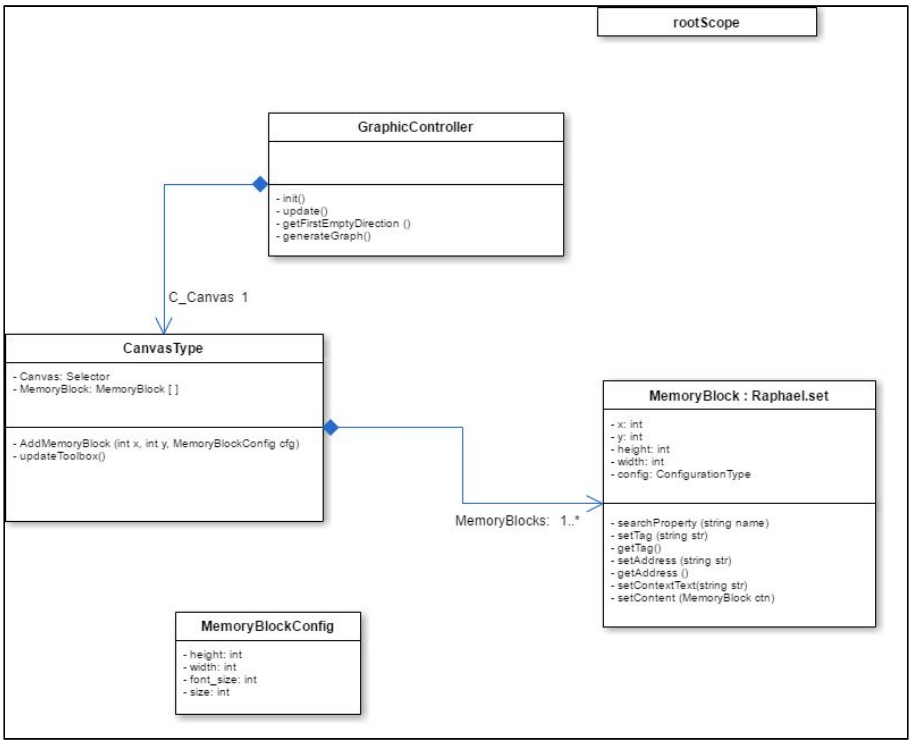



**Figura 4-10: Diagrama de clases módulo gráfico RafaelJS**

### 4.1.2.3 Editor de texto - CKEditor

El módulo del editor de texto es el más sencillo de todos, y queda definido mediante este sencillo diagrama de clases (figura 4-11), que únicamente contiene funciones para establecer/actualizar el contenido definido en el textarea del editor de texto *CKEditor*.

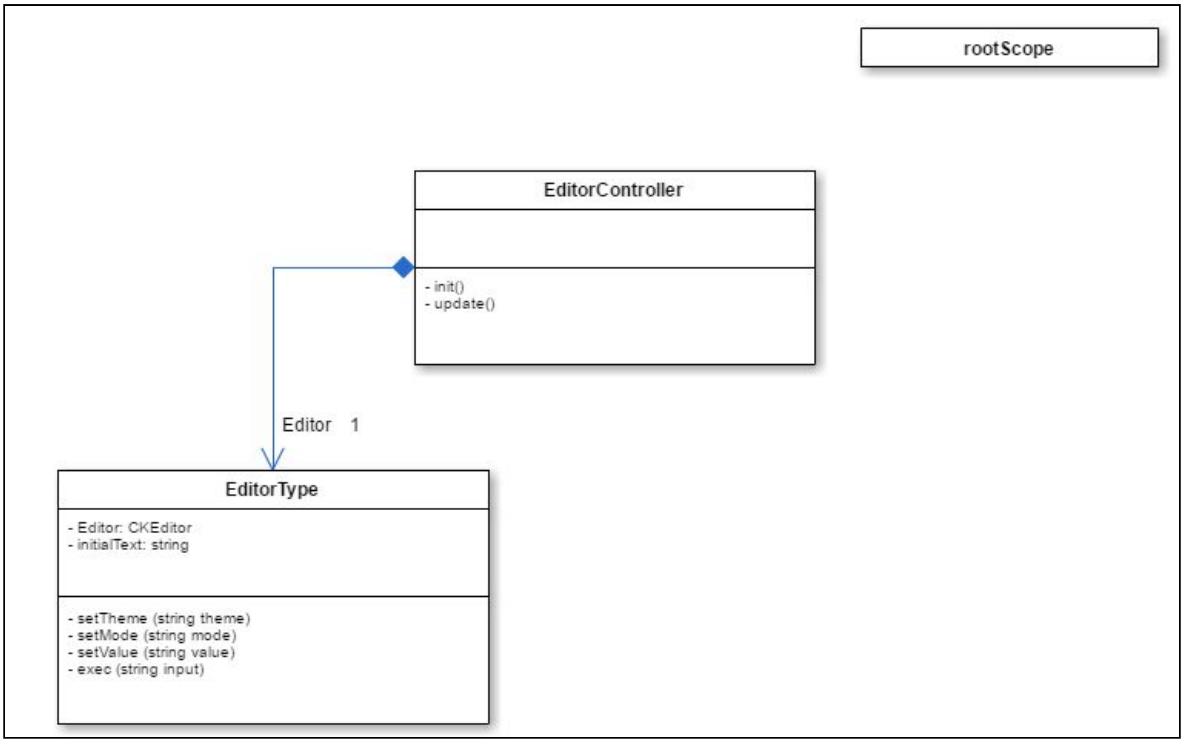



**Figura 4-11: Diagrama de clase módulo CKEditor**

*CKEditor* tiene por defecto un sistema de resaltado de código, teniendo únicamente que definir el modo de programación en el momento en el que inicializas el entorno.

### 4.1.2.4 Módulo principal

Finalmente nos queda el módulo principal, que se define en el **MainController**, y es el encargado de comunicar entre sí los otros 3 sub-módulos (*Blockly*, *CKEditor* y *RafaelJS*). Este módulo es además el que contiene la función principal de procesamiento de las instrucciones, es decir, **el intérprete** que hemos creado para procesar el algoritmo *Blockly*.

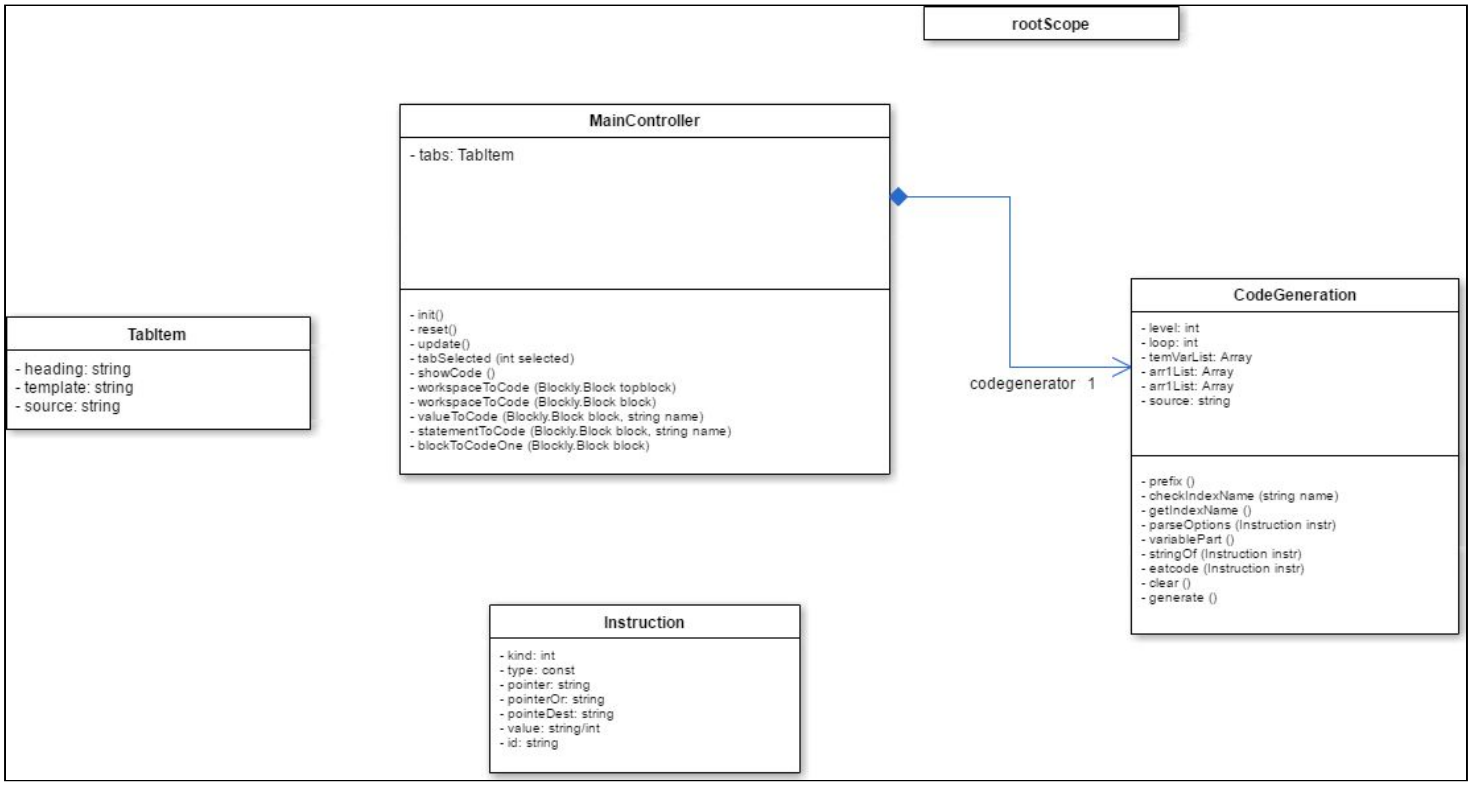



**Figura 4-12: Diagrama de clase módulo Principal**

Tal y como se puede observar en el diagrama de clases de la figura 4-12, este controlador posee un objeto de tipo **CodeGeneration**, que define el conjunto de array y variables necesarios para procesar las instrucciones obtenidas después de haber procesado el algoritmo de *Blockly*, llamando a la función **showCode()**, que, tal y como se explicará en el apartado de desarrollo, se encargará de procesar una a una las instrucciones generando el equivalente código C y correspondiencia gráfica en *RafaelJS*.

Cabe destacar otro tipo de objeto que aparece en el diagrama que es **TabItem**. Tal y como se especifica en el conjunto de requisitos funcionales de la aplicación, concretamente el requisito EXTENSIBILIDAD_RFB_4, nuestro proyecto debe permitir especificar nuevos contextos que incluyan bloques que añadamos externamente (definiendo la vista y modelo del nuevo bloque en un javascript ajeno al proyecto).

Por ello, tal y como aparece reflejado en las maquetas del apartado 4.2, nuestro algoritmo *Blockly* estará contenido en una pestaña que determinará el contexto, es decir, el conjunto de bloques (definidos mediante el toolbox) a los que tiene acceso, incluyendo los bloques básicos y los nuevos bloques posibles añadidos dinámicamente.

Cada **TabItem** tendrá asociado un *heading* (título del contexto), un *template* (donde se especifica el nuevo fragmento del toolbox que se va a añadir) y un *source* (que define la vista y modelo de los nuevos bloques a añadir). De esta forma, podemos crear en el futuro por ejemplo un contexto llamado *'TADS'*, donde, además de los bloques fundamentales, podemos incluir bloques nuevos para definir en el algoritmo *Blockly* tipos abstractos de datos.

## 4.2 Diseño de la interfaz

A la hora de diseñar la interfaz, quisimos mantener un estilo simple que permitiese interactuar con la aplicación directamente, sin ningún tipo de control de acceso.

Empezamos el diseño pensando en mostrar todas las pantallas a la vez, Figura 4-13, de forma que los tres módulos se viesen a primera vista para poder ver cómo se afectan unos entre otros.

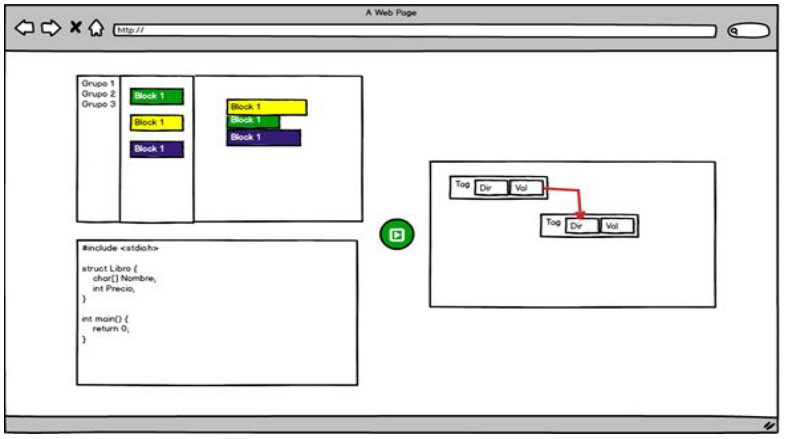



**Figura 4-13: Mockup inicial**

En seguida nos dimos cuenta de que el problema era el espacio, y al querer intentar mostrar tanta información en una sola pantalla, no sería posible ver todos los módulos a la vez. Por ello decidimos incluir un radio button que permitiese alternar entre la vista del **módulo Blockly** y el **módulo CKEditor** (Figura 4-14).

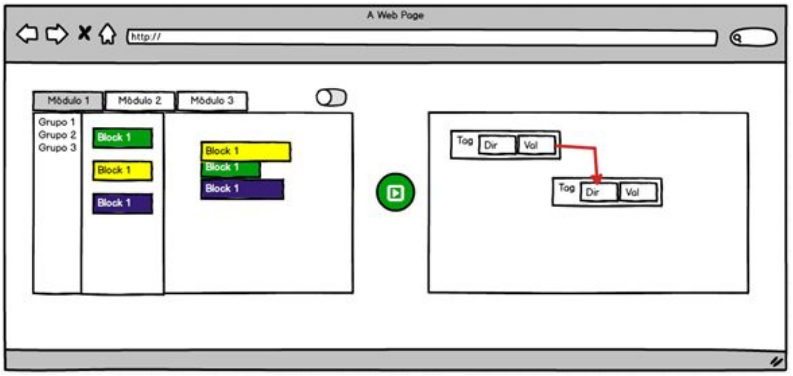



**Figura 4-14: Maqueta con radio button (blockly)**

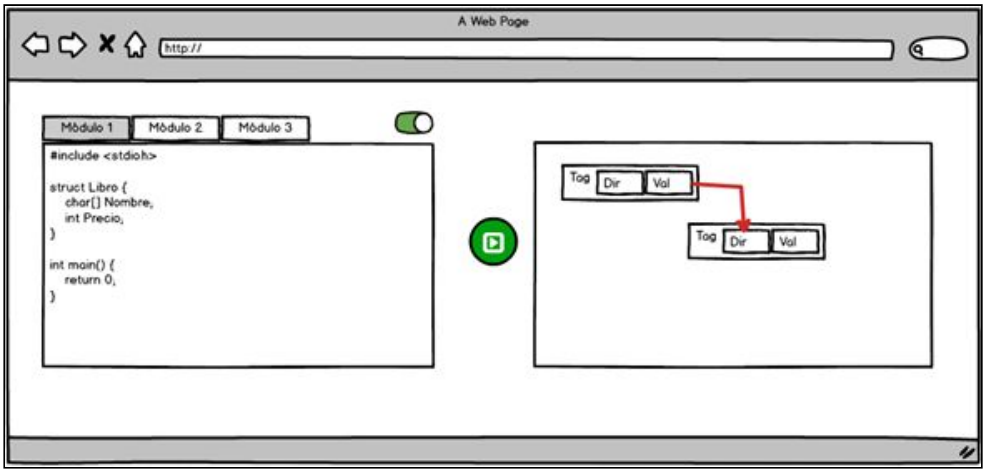



**Figura 4-15: Maqueta con radio button (ckeditor)**

La maqueta de la imagen 4-15 corresponde con la vista alternativa del panel izquierdo una vez pulsamos el radio button, haciendo que se muestre el área de texto CKEditor.

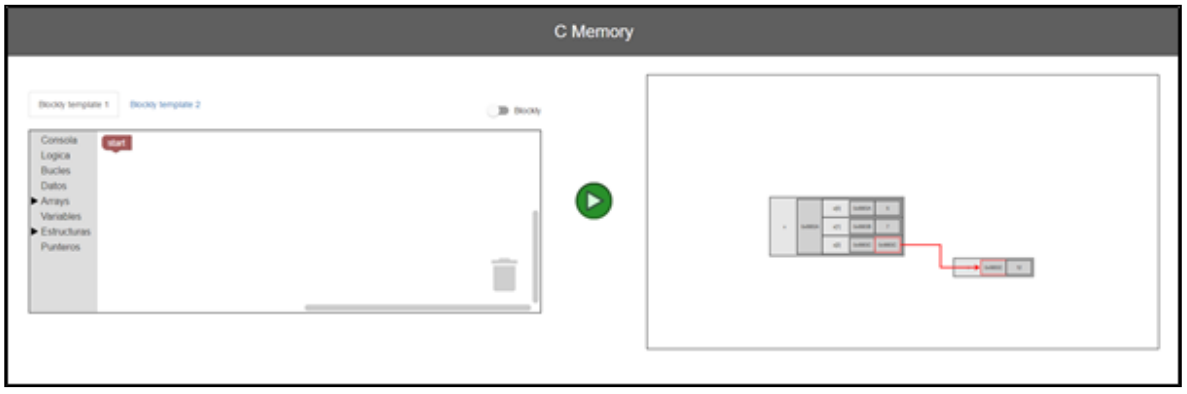



**Figura 4-16: Interfaz final**

La figura 4-16 se corresponde con el diseño final de la aplicación, conseguida con la combinación de html, css y bootstrap. En la imagen se pueden diferenciar el módulo de *Blockly* a la izquierda, y el módulo del svg *RafaelJS* a la derecha, cuyo contenido se actualiza cada vez que damos al botón central verde de ejecución del algoritmo.

También se puede observar arriba a la derecha del módulo *Blockly* un pequeño radio button, que será el encargado de alternar la vista del panel izquierdo entre *CKEditor* (con el contenido del algoritmo definido en el lenguaje de bloques en su equivalente en C) y *Blockly*.

# 5 Desarrollo

En esta sección se explica cómo se ha llevado a cabo la implementación de los 3 sub-módulos de este TFG en el proyecto (*Blockly*, *CKEditor* y *RafaelJS*) y uno general. Se expone el proceso de desarrollo del modelo de datos que se ha usado para manejar la información.

## *5.1 Modelo de datos*

### 5.1.1 Módulo Blockly

El primer paso para empezar a desarrollar en Blockly es definir la estructura de bloques que se va a utilizar, es decir, las unidades atómicas que se utilizarán para entrelazarlas entre sí.

Al tratarse este proyecto sobre la gestión de memoria dinámica en C, lógicamente estas unidades se corresponderán con las funciones básicas de este lenguaje, pudiendo destacar las **funciones de definición** (declarar variables, punteros y estructuras) y **funciones de operación** (bucles, asignaciones, comparaciones...).

#### 5.1.1.1 Definición de estructura - Toolbox

Sabiendo el tipo de operaciones que se tienen que definir en *Blockly*, lo primero que tuvimos que hacer fue diseñar una estructura de árbol que contendría todas las secciones necesarias para englobar las distintas **operaciones de gestión de memoria dinámica**.

| | | |
|---|---|---|
| Consola | Funciones para imprimir resultados | |
| Lógica | Funciones condicionales | |
| Bucles | Funciones de definición de bucles | |
| Datos | Datos numéricos y operaciones aritméticas | |
| Arrays | Definición e instanciación de arrays | |
| Variables | Definición e instanciación de variables (string \| int) | |
| Estructuras | Definición | Definición de estructura y campos |
| | Instanciación | Instanciación de estructuras definidas |
| Punteros | Creación, asignación y reserva de memoria para punteros | |

**Figura 5-1: Estructura de toolbox Blockly**

Una vez definida la estructura (tal y como aparece en la figura 5-1), codificamos la representación correspondiente en xml para *Blockly*, definiendo así el **Toolbox.**

```
<xml id="toolbox" style="display: none">
    <category name="Consola">
        <block type="webc_print_num"></block>
        <block type="webc_print_str">
            <value name="TEXT">
                <block type="text" movable="false"></block>
            </value>
        </block>
    </category>
    <category name="Logica">
        <block type="webc_if"></block>
        <block type="webc_ifElse"></block>
        <block type="webc_logic_compare"></block>
        <block type="webc_logic_operation"></block>
        <block type="webc_logic_boolean"></block>
    </category>
    <category name="Bucles">
        <block type="webc_while"></block>
        <block type="webc_for">
            <value name="FROM">
                <block type="webc_math_number" movable="false" editable="false" deletable="false">
                    <field name="NUM">0</field>
                </block>
            </value>
            <value name="TO">
                <block type="webc_math_number">
                    <field name="NUM">10</field>
                </block>
            </value>
        </block>
    </category>
    <category name="Datos">
        <block type="webc_math_number"></block>
        <block type="webc_math_arithmetic"></block>
        <block type="webc_null"></block>
    </category>
    <category name="Arrays">
        <category name="[ ]">
            <block type="webc_array_one_resize_"></block>
            <block type="webc_array_one_get"></block>
            <block type="webc_array_one_set"></block>
        </category>
        <category name="[ ][ ]">
            <block type="webc_array_two_resize"></block>
            <block type="webc_array_two_get"></block>
            <block type="webc_array_two_set"></block>
        </category>
    </category>
    <category name="Variables">
        <block type="webc_variables_set"></block>
        <block type="webc_variables_get"></block>
    </category>
    <category name="Estructuras">
        <category name="Definicion">
            <block type="webc_struct_set">
                <value name="NAME">
                    <block type="text" movable="false"></block>
                </value>
            </block>
            <block type="webc_struct_addField"></block>
        </category>
        <category name="Instanciación">
            <block type="webc_struct_create"></block>
        </category>
    </category>
    <category name="Punteros">
        <block type="webc_pointers_set"></block>
        <block type="webc_pointer_change"></block>
        <block type="webc_malloc"></block>
        <block type="webc_free"></block>
    </category>
</xml>
```

**Figura 5-2: Definición de toolbox Blockly**

Observamos que la misma organización en sub-apartados está presente en el xml, donde "**category**" define un conjunto y **"block"** las unidades operacionales de Blockly.

Tal y como se especificó en el apartado de diseño de *Blockly*, los bloques a los que el *toolbox* refiere son aquellos que nosotros debimos de haber definido con anterioridad, codificando su **vista** y **modelo**. Como no es viable mostrar el código de todos estos bloques, referimos en **Anexo C** la abstracción del funcionamiento de los mismos para cada uno de los bloques existentes (utilizamos el formato que se explica en el apartado 4.1.2.1).

### 5.1.1.2 Definición de bloques

Tal y como se refleja en el **Anexo C**, definimos no solo el comportamiento gráfico de los bloques entre sí (posibles conexiones, entradas, colores...) sino también el tipo de instrucción que generan dentro del algoritmo, que posteriormente será procesada por el intérprete del módulo principal generando así su equivalente en **lenguaje C** y en elementos gráficos por bloques de memoria en *RafaelJS*.

Hay un bloque que no aparece referido en la tabla, que es el **bloque start** (figura 5-3), que aparece inicialmente incorporado al algoritmo *Blockly* y se encarga de especificar

el conjunto de bloques que se van a procesar, pues todos aquellos que no estén conectados a éste directa o indirectamente, no se considerarán a la hora de procesar el algoritmo (a excepción de los bloques de definición de estructura, tal y como se explica más adelante).


**Figura 5-3: Bloque start**

De todo este conjunto de bloques, la mayoría definen su modelo únicamente con la instrucción que generan a excepción de los bloques que refieren a **estructuras**. Este tipo de bloques son especiales porque si definición afecta a otros bloques, concretamente a aquellos que definen variables y punteros, pues, al crear una estructura, esta debe de estar disponible dinámicamente para los nuevos bloques antes de ejecutar el algoritmo (es decir, procesarlo con el intérprete).

Para ello hicimos uso de los **eventos *Blockly***, que se disparan cada vez que detectan cambios como creación, modificación y borrado de bloques.

```
onchange: function(event) {
    this.blockName = event.newValue;
    if (event.name == 'TEXT') {
        var stIndex = arrayObjectIndexOf($rootScope.Webc.allStructures, 'Tag', event.oldValue);
        if (stIndex == -1) {
            var varObj = {
                Type: CONST_STRUCT,
                Tag: event.newValue,
                Valor: []
            };
            $rootScope.Webc.allStructures.push(varObj);
        } else {
            $rootScope.Webc.allStructures[stIndex].Tag = event.newValue;
        }
    }

    if(event.newInputName == "STRUCT" || event.oldInputName == "STRUCT" || true){
        var stIndex = arrayObjectIndexOf($rootScope.Webc.allStructures, 'Tag', $rootScope.Webc.valueToCode(this, 'NAME'));
        if (stIndex != -1) {
            $rootScope.Webc.allStructures[stIndex].Valor = [];
        }
        $rootScope.Webc.onStructureName = $rootScope.Webc.valueToCode(this, 'NAME');
        $rootScope.Webc.statementToCode(this, 'STRUCT');
    }

},
```

**Figura 5-4: Control de evento *Blockly* onchange()**

En la figura 5-4 se muestra el *handler* del modelo del bloque que se encarga de procesar los cambios del módulo *Blockly* para que, en función de los bloques de tipo **definición de estructura** que haya instanciados en el algoritmo, se actualicen en el conjunto de tipos de variables, estando así disponible para el resto de bloques.

**IMPORTANTE:** Durante el desarrollo, nos dimos cuenta de que los eventos de cambio *onDelete* no se disparaban correctamente al eliminar un bloque (pues en caso de eliminar un bloque de tipo estructura, teníamos que deshabilitar su tipo para el resto de bloques), de forma que tuvimos que modificar la librería Blockly, que es de código abierto, para procesar este tipo de eventos correctamente (lo hicimos asociando un nuevo *handler* a los bloques llamado *onDelete()*, tal y como aparece en la figura 5-5).

```
onDelete: function() {
    var stIndex = arrayObjectIndexOf($rootScope.Webc.allStructures, 'Tag', this.blockName);
    if (stIndex != -1) {
        $rootScope.Webc.allStructures.splice(stIndex, 1);
    }
}
```

**Figura 5-5: Control de evento *Blockly* onDelete()**

Tal y como se aprecia en la tabla 5-1, el bloque de definición de estructura 'web_struct_set' es unos de los pocos bloques de definición que no admiten conexiones a bloques anteriores ni posteriores, y es porque este tipo de bloques se procesan aún no estando conectados con el bloque de inicio **start** (figura 5-6).


**Figura 5-6: Bloque de definición de estructura que se considera dentro del algoritmo**

### 5.1.2 Módulo CKEditor

Este módulo es el más sencillo ya que no tiene que ser interactivo, sino que solo lo necesitamos para representar el código C correspondiente a los datos representados en Blockly con el marcado de código correspondiente.

```
1  #include <stdio.h>
2      int main() {
3          return 0;
4      }
```

**Figura 5-7: Función main C**

Lo único que hay que hacer es instanciar el editor, establecerlo en modo **lenguaje C** y actualizar el contenido dinámicamente en función de los bloques definidos en Blockly.

```
<div ng-show="showTextEditor" id="interpreterArea" class="interpreter" ng-controller="EditorController">
    <div id="editor">
    </div>
</div>
```

**Figura 5-8: Función main C**

Una vez declaramos el área en el que se representará el editor, lo inicializamos en el controlador correspondiente de AngularJS “EditorController”, estableciendo el módo de resaltado para que sea el del lenguaje C llamando a la función **setMode()**, tal y como se muestra en la figura 5-9.

```
$rootScope.C_Editor = {};

$rootScope.C_Editor.init = function() {
$rootScope.C_Editor.editor = ace.edit("editor");
$rootScope.C_Editor.editor.setTheme("ace/theme/chrome");
$rootScope.C_Editor.editor.getSession().setMode("ace/mode/c_cpp");
$rootScope.C_Editor.editor.$blockScrolling = Infinity;
```

**Figura 5-9: Definición del editor de texto CKEditor**

### 5.1.3 Módulo RaphaelJS

Para utilizar esta librería primero definimos el área en el que se va a representar el svg (figura 5-10).

```
<div class="graphic" ng-controller="GraphicController">
    <div id="c_canvas"></div>
</div>
```

**Figura 5-10: Div html de módulo gráfico**

Una vez creada en área donde se va a dibujar, inicializamos un objeto de tipo Raphael utilizando la id del "div" (figura 5-11).

```
$rootScope.C_Canvas.R = Raphael("c_canvas");
$rootScope.C_Canvas.R.setSize($rootScope.C_Canvas.Canvas.width(), $rootScope.C_Canvas.Canvas.height());
```

**Figura 5-11: Definición de svg RafaelJS**

A partir de aquí, definimos el conjunto de estructuras y funciones necesarias para establecer la funcionalidad básica del programa.

#### 5.1.3.1 Comunicación con Blockly

Tal y como hemos explicado antes, el módulo de Blockly se encarga de generar las instrucciones, variables, punteros y estructuras necesarias para gestionar los datos.

Este módulo provee de una función llamada *generateGraph()* que se encargará de actualizar los elementos del svg acorde a los datos almacenados mediante los módulos de Blockly.

Primero tuvimos que definir las funciones que se encargan de representar los datos tal y como se ha explicado en el apartado **4.1.2.2**, donde un único bloque de memoria consiste en un conjunto de rectángulos y texto que agruparemos en un Raphael.set (figura 5-12).

```
var memoryContent = r.set();

var generalRect = r.rect(x, y, width, height);
generalRect.name = "generalRect";
var tag = r.text(x + usableWidth / 6 + config.padding, y + usableHeight / 2 + config.padding, "TAG");
tag.name = "tag";
var addressRect = r.rect(x + usableWidth / 3 + config.padding, y + config.padding, usableWidth / 3, usableHeight);
addressRect.name = "addressRect";
var addressText = r.text(x + usableWidth * 3 / 6 + config.padding, y + usableHeight / 2 + config.padding, "DIR");
addressText.name = "addressText";
var contentRect = r.rect(x + usableWidth * 2 / 3 + config.padding, y + config.padding, usableWidth / 3, usableHeight);
contentRect.name = "contentRect";
var contentSet = r.set();
contentSet.name = "contentSet";

generalRect.attr({ fill: "#E6E6E6", stroke: "black", "fill-opacity": 1, "stroke-width": 1, cursor: "move" });
addressRect.attr({ fill: "#D0D0D0", stroke: "black", "fill-opacity": 1, "stroke-width": 1, cursor: "move" });
contentRect.attr({ fill: "#D0D0D0", stroke: "black", "fill-opacity": 1, "stroke-width": 1, cursor: "move" });

addressText.attr({ "font-size": cfg.font_size });
tag.attr({ "font-size": cfg.font_size });

memoryContent.push(
        generalRect,
        tag,
        addressRect,
        addressText,
        contentRect,
        contentSet
    );
```

**Figura 5-12: Definición de svg RafaelJS**

El tamaño de los cuadrados y el tamaño de fuente se definen mediante los parámetros de la variable de configuración que se pase (figura 5-13).

```
var memoryConfig = {
    height: 30,
    width: 130,
    font_size: 8,
    padding: 3
};
```

**Figura 5-13: Variable de configuración bloque de memoria**

Asímismo, estos bloques tienen un conjunto de funciones que nos permiten editar los valores de sus atributos tag, dirección y valor así como la posibilidad de incluir bloques dentro del contenido de otro.

```
/*Functions*/
memoryBlock.searchProperty = function (name) {
    for(var i = 0; i < this[0].length; i++){
        if(this[0][i].name == name){
            return this[0][i];
        }
    }
    return null;
};

memoryBlock.setTag = function (str) {
    var e = this.searchProperty("tag");
    e.attr({ text: str });
    return this;
};
memoryBlock.getTag = function () {
    var e = this.searchProperty("tag");
    return e.attr("text");
};

memoryBlock.setAddress = function (str) {
    var e = this.searchProperty("addressText");
    e.attr({ text: str });
    return this;
};
memoryBlock.getAddress = function () {
    var e = this.searchProperty("addressText");
    return e.attr("text");
};

memoryBlock.setContentText = function (str) {
    var gr = this.searchProperty("generalRect");
    var t = this.searchProperty("tag");
    var ar = this.searchProperty("addressRect");
    var at = this.searchProperty("addressText");
    var ct = this.searchProperty("contentRect");
    var s = this.searchProperty("contentSet");

    var cfg = memoryBlock.config;

    var contentText = r.text(ct.attr("x") + ct.attr("width") / 2, ct.attr("y") + ct.attr("height") / 2, str);
    contentText.attr({ "font-size": cfg.font_size });

    s.push(contentText);
    this.undrag();
    this.draggable();
    return this;
};
```

**Figura 5-14: Definición de elemento gráfico bloque de memoria**

En cuanto a establecer el contenido, tenemos 2 funciones distintas, *setContentText()*, definida en la figura 5-14, que recibe una string que es la que pone en el contenido del bloque de memoria, y *setContent()*, definida en la figura 5-15, que recibe como parámetro otro bloque, teniendo que calcular el tamaño de éste para poder ajustarlo al contenedor.

```
memoryBlock.setContent = function (ctn) {
    var gr = this.searchProperty("generalRect");
    var t = this.searchProperty("tag");
    var ar = this.searchProperty("addressRect");
    var at = this.searchProperty("addressText");
    var ct = this.searchProperty("contentRect");
    var s = this.searchProperty("contentSet");

    var contentHeight = ct.attr("height");
    var contentWidth = ct.attr("width");

    var usedHeight = ctn.height;

    for (var i = 0; i < s.length; i++){
        usedHeight += s[i].height;
    }

    var dh = usedHeight - contentHeight;
    var dw = ctn.width - contentWidth;

    if(dh > 0){
        this.height = this.height + dh;
        this.width = this.width + dw;

        gr.attr({ width: gr.attr("width") + dw, height: gr.attr("height") + dh });
        ar.attr({ height: ar.attr("height") + dh });
        ct.attr({ width: ct.attr("width") + dw, height: ct.attr("height") + dh });
        t.attr({ y: t.attr("y") + dh/2 });
        at.attr({ y: at.attr("y") + dh/2 });
    }

    //ctn.transform('t230,265s3');
    var lstHeight = 0;
    for (var i = 0; i < s.length; i++){
        lstHeight += s[i].height;
    }

    var dx = ct.getBBox().x - ctn.getBBox().x;
    var dy = (ct.getBBox().y + lstHeight) - ctn.getBBox().y;
    ctn.translate(dx, dy);

    s.push(ctn);
    this.undrag();
    this.draggable();
    return this;
};
```

**Figura 5-15: Definición de función setContent() de elemento gráfico bloque de memoria**

Ahora, para entender mejor el flujo de procesamiento que se sigue para representar los datos de los bloques que se definen con Blockly, supongamos que creamos un puntero de tipo entero p_1, y una variable de tipo entero tring v_1, haciendo que éste primero apunte al segundo (figura 5-16).

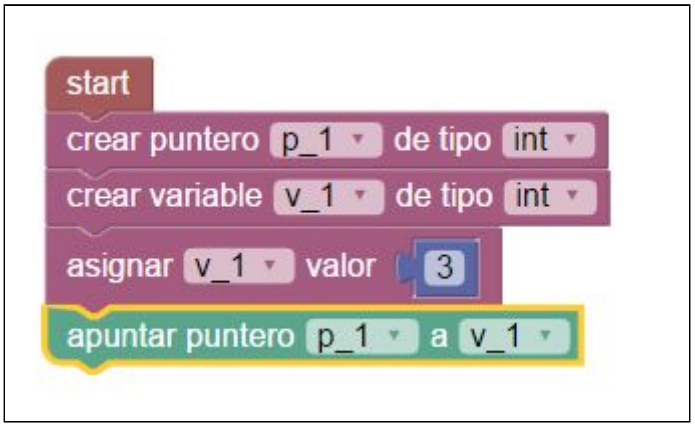



**Figura 5-16: Definición en bloques de ejemplo variable-puntero**

Al crear el algoritmo *Blockly* que aparece reflejado en la figura 5-16, cuando pulsemos el botón de ejecutar de la interfaz, se interpretarán las funciones pertinentes del intérprete creándose el conjunto de variables y punteros que definen al algoritmo. En el caso particular del módulo *RafaelJS*, una vez haya finalizado el intérprete con las instrucciones, llamará a la función *generateGraph()* definida en la imagen 5-17, que creará el conjunto pertinente de bloques de memoria para representar los bloques gráficamente.

```
$rootScope.C_Canvas.generateGraph = function () {

    rootScope.C_Canvas.R.clear();

    var initX = 0;
    var initY = 0;

    var allBlocks = [];

    for (var i = 0; i < $rootScope.C_Canvas.availableMemoryDirections.length; i++) {
        $rootScope.C_Canvas.availableMemoryDirections[i]['empty'] = true;
    };

    /*PINTAR VARIABLES*/
    for (var i = 0; i < $rootScope.Webc.allVariables.length; i++) {
        var item = $rootScope.Webc.allVariables[i];
        var memoryBlock_Item = $scope.AddMemoryBlock(initX, initY, memoryConfig);
        initX += 100;
        initY += 100;
        memoryBlock_Item.setTag(item.Tag);
        var varDir = $rootScope.C_Canvas.getFirstEmptyDirection();
        memoryBlock_Item.setAddress("0x" + varDir);
        // CHECK SI ESTRUCTURA
        if(arrayObjectIndexOf($rootScope.Webc.allStructures, 'Tag', item.Type) != -1){
            if (item.Valor != undefined) {
                for (var m = 0; m < item.Valor.length; m++) {
                    var item_member = item.Valor[m];
                    var memoryBlock_Member_Item = $scope.AddMemoryBlock(initX, initY, memoryConfig);
                    memoryBlock_Member_Item.setTag(item_member.Tag);
                    memoryBlock_Member_Item.setAddress("0x" + varDir);
                    memoryBlock_Member_Item.setContentText(item_member.Valor);
                    memoryBlock_Item.setContent(memoryBlock_Member_Item);
                }
            }
        }else{
             // ES VARIABLE
            memoryBlock_Item.setContentText(item.Valor);
        }
        allBlocks.push(memoryBlock_Item);
    }
```

**Figura 5-17: Función generateGraph() de módulo gráfico**

```
/*PINTAR PUNTEROS*/
for (var i = 0; i < $rootScope.Webc.allPointers.length; i++) {
    var item = $rootScope.Webc.allPointers[i];
    var memoryBlock_Item = $scope.AddMemoryBlock(initX, initY, memoryConfig);
    initX += 50;
    initY += 50;
    memoryBlock_Item.setTag(item.Tag);
    memoryBlock_Item.setAddress("0x" + $rootScope.C_Canvas.getFirstEmptyDirection());
    allBlocks.push(memoryBlock_Item);
}

for (var i = 0; i < $rootScope.Webc.allPointers.length; i++) {

    var item = $rootScope.Webc.allPointers[i];
    var memoryBlock_Item;

    for (var b = 0; b < allBlocks.length; b++){
        if (allBlocks[b].getTag() == item.Tag) {
            memoryBlock_Item = allBlocks[b];
            break;
        }
    }


    if (item.Valor != null) {
        for (var b = 0; b < allBlocks.length; b++){
            if (allBlocks[b].getTag() == item.Valor) {
                memoryBlock_Item.setContentText(allBlocks[b].getAddress());
                $rootScope.C_Canvas.Data.connections.push($rootScope.C_Canvas.R.connection(memoryBlock_Item.searchProperty("contentRect"), allBlocks[b].searchProperty("addressRect"), "red"));
            }
        }
    }
}
```

**Figura 5-18: Función generateGraph() de módulo gráfico. Apartado punteros**

En *RafaelJS* primero tenemos que crear los bloques correspondientes a las variables y a los punteros (tal y como se puede apreciar en la figura 5-18), para finalmente crear las conexiones pertinentes especificando las posibles referencias de punteros-dirección, tal y como se ve en la imágen de ejemplo 5-19.

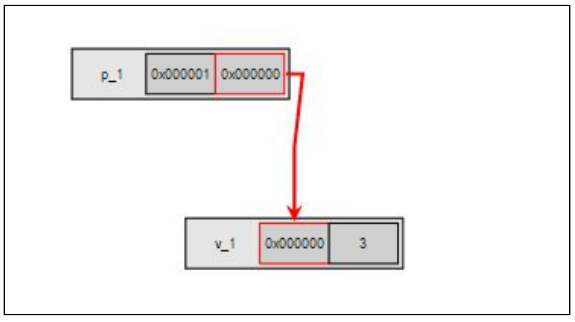



**Figura 5-19: Ejemplo RafaelJS puntero-variable**

### 5.1.4 Módulo General

El módulo principal es el encargado de interconectar cada uno de los sub-módulos y es además el que contiene el **intérprete**, es decir, el encargado de procesar cada una de las instrucciones que se generan mediante Blockly transformándolas en su equivalente a **código C.**

Tal y como se describe en la sección 4.1.4.1, las principales funciones que definen la funcionalidad básica son **showCode()** y las funciones del objeto de tipo **CodeGeneration**.

Utilizaremos el siguiente ejemplo para explicar el flujo de procesamiento que se sigue con cada uno de los bloques para su transformación en código C y visualización equivalente en bloques de memoria.

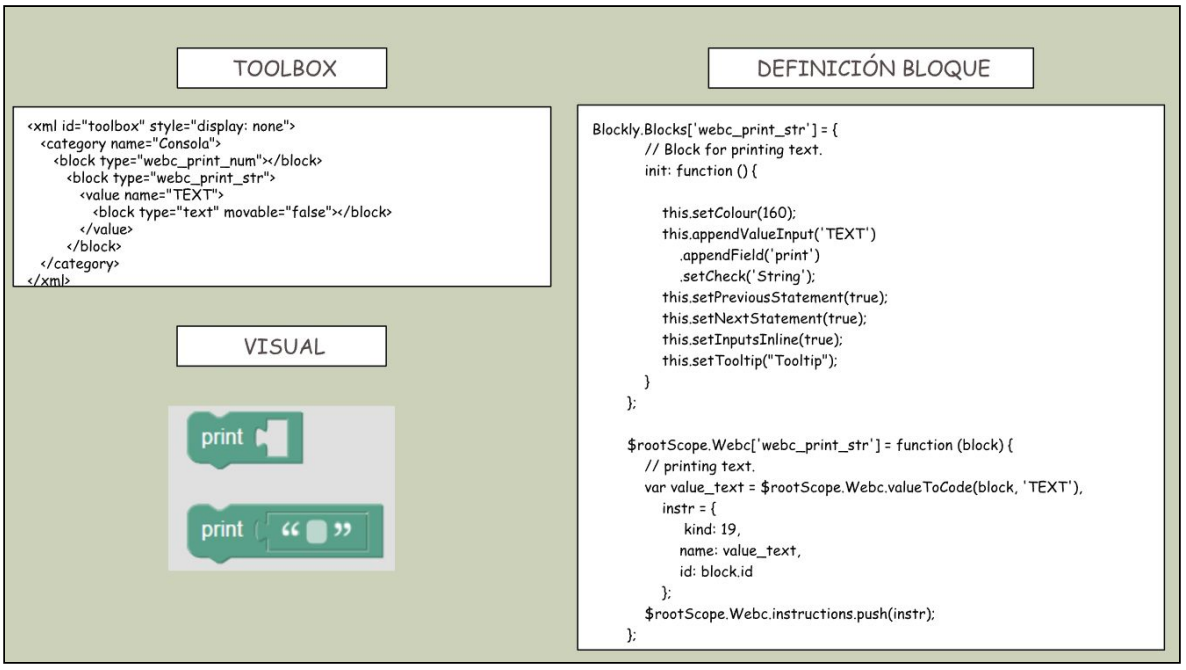



**Figura 5-20: Definición de un nuevo bloque**

Una vez que se ha definido un bloque tal y como se explica en el **apartado 5.1.1**, la función del módulo principal que inicia el proceso de transformación es **showCode()** (que se ejecuta cada vez que se pulsa un botón)**,** que se encargará de procesar las instrucciones generadas por los bloques instanciados.

```
$rootScope.Webc.showCode = function() {

    $rootScope.Webc.funMap = {};

    $rootScope.Webc.instructions = [];
    $rootScope.Webc.workspaceToCode($rootScope.Webc.startblock);

    cg = new CodeGeneration();
    code = cg.generate();

    var editorText = code.trim();
    $rootScope.C_Editor.editor.setValue(editorText);

    $rootScope.C_Canvas.update();

    $.ajax({
        url: 'php/c_compile.php',
        type: 'post',
        data: {
            "instruction": editorText
        },
        success: function(response) {
            //alert(response); RESULTADO COMPILACION
        }
    });

    //** DRAW GRAPH **//
    $rootScope.C_Canvas.generateGraph();

}
```

**Figura 5-21: Función showCode() de módulo Principal**

Lo primero que se hace es llamar a la función **worspaceToCode()** (figura 5-21)**,** a la que se le pasa el puntero al bloque **main** (el bloque inicial) a raíz del cual se van a leer bloques subsiguientes, es decir, los que están adheridos éste (tal y como se puede ver en la figura 5-22).

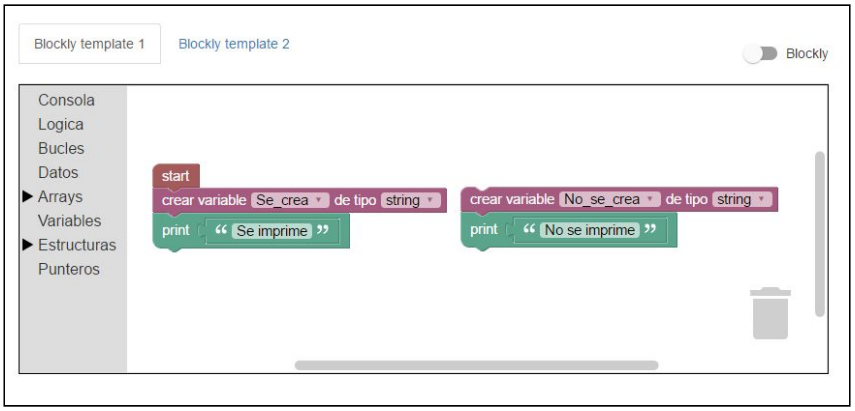



**Figura 5-22: Figura que muestra bloques que se ejecutan**

A raíz de aquí, se recorre en bucle cada uno de los bloques hijos de 'web_start', ejecutando la función de generación de código asociada a cada uno de ellos, que es la encargada de generar y almacenar la instrucción correspondiente en el array de instrucciones **instructions.**

```
// code generation from workspace
$rootScope.Webc.workspaceToCode = function(topblock) {
    var blocks,
        x,
        block,
        code,
        name,
        types = ['procedures_defnoreturn', 'procedures_defreturn'],
        funToCode = function(fname, block) {
            while (block) {
                $rootScope.Webc.blockToCode(block);
                block = block.getNextBlock();
            }
            $rootScope.Webc.funMap[fname] = {
                instructions: $rootScope.Webc.instructions,
                allVariables: $rootScope.Webc.allVariables,
                allPointers: $rootScope.Webc.allPointers,
                varList: null,
                allList: null,
                varMap: null
            };
        };

    blocks = Blockly.mainWorkspace.getTopBlocks(true);
    for (x = 0; block = blocks[x]; x += 1) {

        if (block === topblock) {

            name = '_main_';
            funToCode(name, block);

        } else if (types.indexOf(block.type) >= 0) {

            name = block.getFieldValue('NAME');
            funToCode(name, block);

        } else if (block.type === 'webc_struct_set') {

            name = block.getFieldValue('NAME');
            funToCode(name, block);
        }
    }

    $rootScope.Webc.instructions.push({
        kind: 41 // Intrucción final
    });
};
```

**Figura 5-24: Función workspaceToCode() de módulo principal**

En este caso particular, después de que se ejecute la función de *workspaceToCode()* (figura 5-24), tendremos en el array de $rootScope.Webc.instructions las 3 instrucciones correspondientes a: **instrucción inicial**, **instrucción** que hemos creado de imprimir **string** e **instrucción final** (figura 5-25).


**Figura 5-25: Estado de variables ejemplo impresión de pantalla**

Después de haberse generado las instrucciones correspondientes, se crea un objeto de tipo **CodeGeneration** y se llama a la función **generate()** de éste.

```
var i,
    instr,
    lastBlock = null;

$rootScope.Webc.pc += 1;

var blocks = Blockly.mainWorkspace.getAllBlocks();
for (var i = 0; i < blocks.length; i++) { // Limpiamos selecciones
    blocks[i].removeSelect();
}

if ($rootScope.Webc.pc >= $rootScope.Webc.instructions.length - 1) { // Reiniciamos PC en caso de llegar a última instrucción
    $rootScope.Webc.pc = 1;
    this.clear();
}

if (Blockly.mainWorkspace.getBlockById(lastBlock) != null) { // Eliminamos selección de bloque anterior
    Blockly.mainWorkspace.getBlockById(lastBlock).removeSelect();
}
instr = $rootScope.Webc.instructions[$rootScope.Webc.pc];
if (Blockly.mainWorkspace.getBlockById(instr.id) != null) { // Procesamos bloque
    Blockly.mainWorkspace.getBlockById(instr.id).addSelect();
    this.level = $rootScope.Webc.level; // Para tabulación
    this.eatcode(instr); // Generamos código de bloque
    $rootScope.Webc.level = this.level;
    $rootScope.Webc.src += this.source;
    $rootScope.Webc.structures += this.structures; // Control de definición de estructuras
}
```

**Figura 5-26: Gestion de PC en función generate() del objeto CodeGeneration**

Primero controlamos mediante el PC (**$rootScope.Webc.pc**) la instrucción que se ejecuta (pues los bloques se ejecutan de 1 en 1 cada vez que pulsamos el botón) y almacenamos el **$rootScope.Webc.src** el código C que se genera a través de la función **CodeGeneration.eatcode()** (figura 5-26)**.**

```
eatcode: function(instr) {

    if (codeGeneratorAppend(instr, this.source) == true) {
        return;
    }

    if (instr.kind === 19) {
        // print string
        this.source += this.prefix() + 'printf(\"' + instr.name + '\");\n';
    }
```

**Figura 5-27: Función eatcode() de objeto CodeGeneration**

Esta función (figura 5-27) se encarga de, en función de la instrucción que le llegue, devolver la representación en lenguaje C en función de los parámetros de ésta.

En el caso de ejemplo, sencillamente tenemos que pintar el **prefijo**, que corresponde a la cantidad de tabuladores que se controlan con la variable **$rootScope.Webc.level**, el identificador de la función de impresión en C, en este caso printf(), y el valor que queremos imprimir, que se define en el parámetro name de la instrucción.

De esta forma finalizamos el flujo de procesamiento de los bloques Blockly a su representación en código C, quedando solo tener que pintarlo en el editor de texto CKEditor (figura 5-28) y su representación gráfica en el módulo de RafaelJS (que en este caso de ejemplo no procede).


**Figura 5-28: Resultado final en C de bloque de impresión**

# 6 Integración, pruebas y resultados

## *6.1 Pruebas unitarias*

Una vez desarrollado cada módulo de la aplicación, se realizan varias pruebas para comprobar el funcionamiento global correcto del sistema y la correcta integración del conjunto de módulos del proyecto. Se realiza una serie de pruebas sobre cada módulo desarrollado para comprobar su comportamiento antes de pasar a la siguiente fase de desarrollo.

### 6.1.1 Pruebas unitarias en el módulo Blockly

Debido a la falta de información y documentación de esta librería, fue importante comprobar la funcionalidad de cada uno de los bloques así como la interacción entre ellos, por consiguiente, los factores que tuvimos que comprobar fueron:

- Las conexiones entre bloques deben permitirse sólo en aquellos casos en los que tenga sentido en función del contexto, no se deben poder instanciar bloques de asignación dentro de bloques de definición, como por ejemplo el bloque de definición de estructuras.
- Se deben diferenciar bloques de tipo STRING con bloques de tipo INT, de forma que sea un error introducir un input de tipo cadena en un bloque que espera una entrada de tipo entero y viceversa.

### 6.1.2 Pruebas unitarias en el módulo Editor

Este módulo es el más pequeño y sencillo de todos, luego las pruebas necesarias para asegurarnos del correcto funcionamiento no fueron muchas.

- Comprobar que la representación de los bloques Blockly en C aparecen en orden, y que dentro de los bloques de tipo bucle el código contenido no se escribe múltiples veces.
- La definición de las estructuras debe aparecer fuera de la función main(), y las definiciones de las variables deben aparecer antes de cualquier tipo de asignación.

### 6.1.3 Pruebas unitarias en el módulo Gráfico

Este conjunto de pruebas debe verificar que los elementos gráficos son escalables y desplazables, así como que las correspondientes conexiones de los punteros se mantienen y corresponden con la estructura definida en el módulo de Blockly.

- En caso de que dos variables sean distintas, es decir, tengan un tag diferente entre sí, la dirección de éstas no puede ser la misma

- El tamaño de los bloques de memoria debe escalar en dinámicamente en función de los datos de su contenido.
- Los bloques de memoria que hacen referencia a un puntero deben apuntar siempre la dirección del bloque de memoria objetivo, no a su tag ni a su contenido.

## 6.2 Pruebas de integración

Las pruebas de integración tienen como objetivo verificar el correcto funcionamiento de los módulos en los que se divide la aplicación trabajando juntos. A continuación mostramos dos casos donde testeamos la comunicación entre los distintos módulos.

Primero probamos a crear una lista enlazada definiendo en *Blockly* el algoritmo pertinente. El resultado de dicha ekecución es el que se puede ver en la figura 6-1.

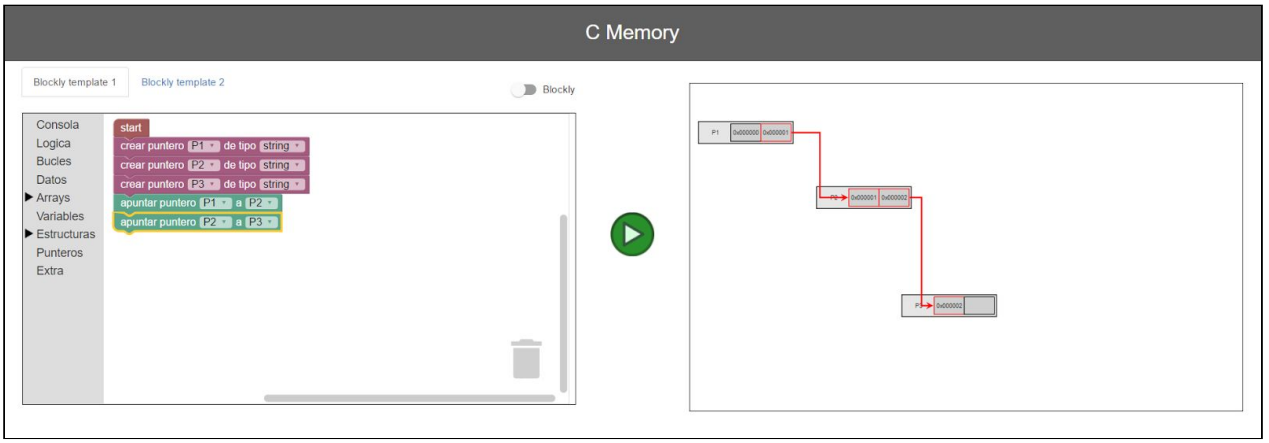



**Figura 6-1: Prueba lista enlazada**

También hay que comprobar el funcionamiento de creación y definición de estructuras del proyecto, dando como resultado gráfico lo que se puede observar en la imagen 6-2.

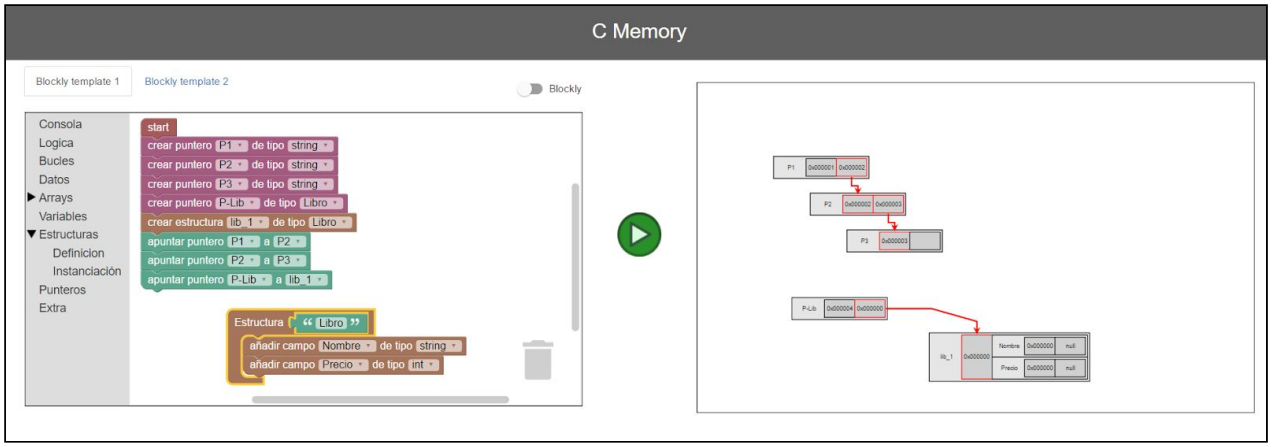



**Figura 6-2: Prueba lista enlazada y estructura**

# 7 Conclusiones y trabajo futuro

## 7.1 Conclusiones

A lo largo del desarrollo de este proyecto se ha estudiado el funcionamiento de diversas tecnologías que, sin estar directamente relacionadas entre sí, se han utilizado juntas para formar un sistema completo y funcional. Mediante la combinación de estas tecnologías se ha construido una herramienta web que permite a los usuarios entender mejor la gestión de memoria dinámica y estudiar y evaluar el comportamiento de distintos tipos de estructuras de datos.

Inicialmente queríamos que el sistema pudiera gestionar aplicaciones complejas multimodulo con tads como las de programación 2. Esto implicaba parte de datos (nuevas estructuras) y parte de funciones. La primera implicó reescribir el intérprete de blockly y decidimos centrarnos en este TFG en la parte de datos. Es por ello que ha habido características que en un principio se querían añadir (tales como funciones y TADS) que no se han podido incluir en la versión final de este TFG, pues suponían un tiempo de desarrollo que excede las competencias de este trabajo. Aún así, hemos conseguido crear una buena estructura que ya sirve para facilitar el aprendizaje de los principios básicos de la gestión de memoria dinámica fácilmente accesible por todo el mundo, pues lo único que se necesita es un navegador.

También nos gustaría comentar que la mayor dificultad con la que nos hemos encontrado a la hora del desarrollo, era con la falta de documentación. Las librerías de *Blockly* y *RafaelJS* nos fueron de mucha utilidad pues, con un pequeño conjunto de funciones básicas pudimos definir toda la funcionalidad del proyecto, no sin antes mucho investigar y darnos de bruces contra la pared cada vez que algo fallaba, pues, al no haber casi documentación online (y al no ser muy popular, tampoco hay mucho debate en los foros), tuvimos casi que solucionar las incidencias a base de prueba y error.

De cualquier forma, esperamos que este proyecto sirva como base e iniciativa para crear uno más amplio y ambicioso, para finalmente generar una herramienta de aprendizaje útil mediante la cual los alumnos puedan iniciarse fácilmente en los conceptos relacionados con la memoria dinámica.

## 7.2 Trabajo futuro

Durante el proceso de diseño e implementación, fueron surgiendo ideas y mejoras aplicables al proyecto que hubo que ir descartando en favor de otras funcionalidades más necesarias para una primera versión de la aplicación. Algunas de estas ampliaciones del sistema son:

- **Inclusión de bloques para definir funciones:** Las funciones suponen un campo importante en el aprendizaje de la gestión de memoria dinámica, pero añadirlos del proyecto habría supuesto la gestión de ámbitos de

variables y de contexto así como una nueva forma de visualización gráfica de los bloques de memoria, diferenciando las variables en función del ámbito.

- **Nuevos módulos de blockly:** A pesar de estar implementada la funcionalidad de escalabilidad, no se han añadido módulos nuevos, con piezas para generar listas enlazadas por ejemplo.

- **Bidireccionalidad código-blockly:** Esta nueva funcionalidad supondría la creación de un intérprete C que transformarse el contenido del editor de texto a su correspondiente representación en *Blockly* y *RaphaelJS*.

- **Ejecución del código:** Una vez se ha definido la estructura *Blockly* y se obtiene la representación en C, en el futuro se podría añadir un apartado que mostrase el resultado de ejecutar ese código en el servidor.

- **Definición de TADS:** Debido al tamaño del proyecto, funcionalidad como la posibilidad de añadir tipos abstractos de datos no se han podido incluir en el algoritmo, quedando así como trabajo futuro para un posible nuevo TFG.

# Referencias


[1] AngularJS. Disponible en: https://angularjs.org/. Último acceso 2017.

[2] Blockly. Disponible en: https://developers.google.com/blockly/. Último acceso 2017.

[3] Visual programming language. 2017; Disponible en: https://en.wikipedia.org/w/index.php?title=Visual_programming_language&oldid=786959079. Último acceso 2017.

[4] Blockly. Disponible en: https://developers.google.com/blockly/, Último acceso 2017.

[5] Bootstrap. Disponible en: http://getbootstrap.com/. Último acceso 2017.

[6] RafaelJS. Disponible en: http://dmitrybaranovskiy.github.io/raphael/.

[7] CKEditor. Disponible en: http://ckeditor.com/, Último acceso 2017.

[8] Ritchie-Dennis-M, Kernighan-Brian-W. The C programming language. 2nd ed. ed. United States: Pearson Prentice Hall; 1988.

[9] Boehm BW. Software cost estimation with Cocomo II. Nachdr. ed. Upper Saddle River, NJ: Prentice Hall; 2009.

[10] w3schools. Disponible en: https://www.w3schools.com/

[11] code academy. Disponible en: https://www.codecademy.com/es

[12] code school. Disponible en: https://www.codeschool.com/

[13] cplusplus. Disponible en: http://www.cplusplus.com/

[14] d3. Disponible en: https://d3js.org/

[15] GoJS. Disponible en: https://gojs.net/latest/index.html

[16] Svg.js. Disponible en: http://svgjs.com/

[17] TinyMCE. Disponible en: https://www.tinymce.com/

[18] oCanvas. Disponible en: http://ocanvas.org/

[19] Three.js. Disponible en: https://threejs.org/

[20] paperjs. Disponible en: http://paperjs.org/

[21] Scratch!. Disponible en: https://scratch.mit.edu/

[22]Snap. Disponible en: http://snap.berkeley.edu/

# Glosario

**AngularJS** — *Framework* de JavaScript para la creación de aplicaciones web con la filosofía SPA. Su objetivo principal es aumentar las aplicaciones del lado Cliente con el patrón MVC reduciendo la carga de trabajo al lado Servidor.

**API** — *Application Programming Interface*. Conjunto de subrutinas, funciones y procedimientos que ofrecen ciertas bibliotecas para ser utilizados por otro software como una capa de asbtracción.

**Controller** — Responde a eventos e invoca peticiones al modelo cuando se hace alguna solicitud sobre la información. También puede enviar comandos a su vista asociada si se solicita un cambio en la forma en que se presenta el modelo. En el caso particular de AngularJS, es un objeto javascript que contiene el conjunto de funciones y eventos que comunican el template con el modelo definido en él.

**Framework** — Estructura de soporte definida, en la cual un proyecto de software puede ser organizado y desarrollado en una plataforma o marco de trabajo.

**HTML** — *Hyper Text Markup Language*. Lenguaje de marcado de contenidos en páginas web basado en etiquetas.

**JavaScript** — Lenguaje de programación interpretado que se utiliza para crear páginas web dinámicas, es decir, que incorporan efectos y/o acciones.

**Responsive** — Diseño web adaptable cuyo objetivo es adaptar las páginas web al dispositivo que las esté visualizando.

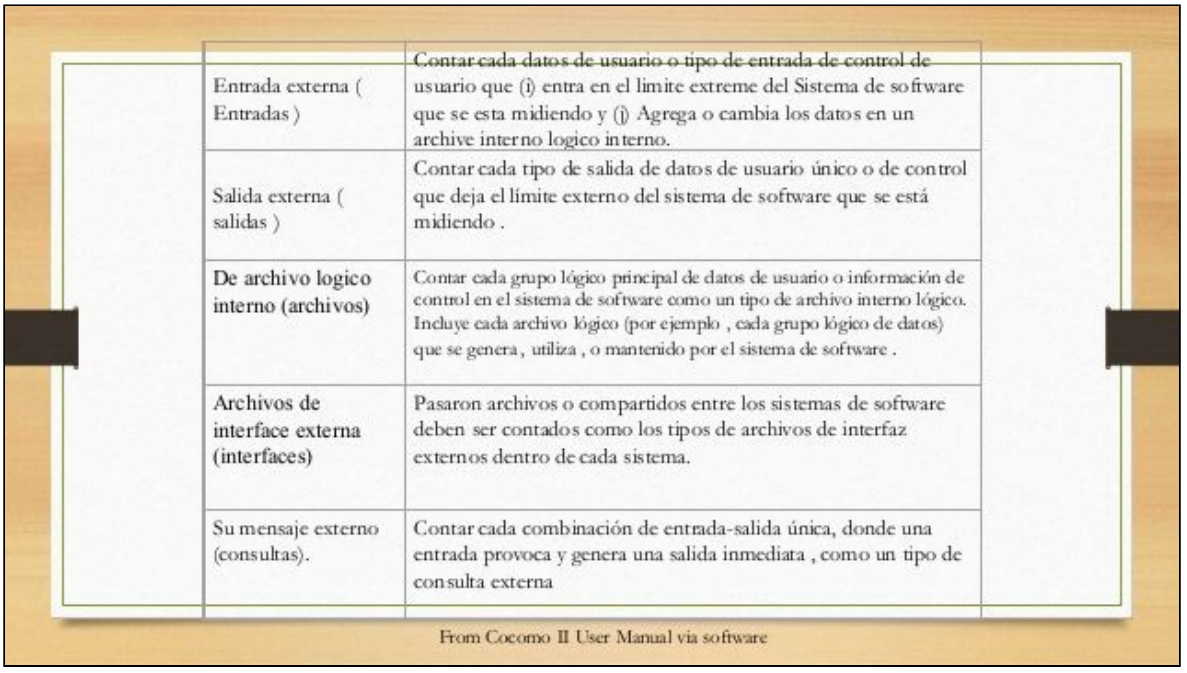
**Scope** — Contexto de un *Controller* en Angular. Contiene una serie de atributos para manejar los datos entre la vista y el controlador.

**SpaguettiCode** — Programa informático con código desordenado, complejo e incomprensible.

**Toolbox** — Menú definido en xml que contiene el conjunto de bloques que el usuario puede crear para definir el algoritmo. Su estructura se define como un árbol de nodos en el que se pueden definir subconjuntos de categorías de bloques.

# Anexos

## A. Manual COCOMO II: Tipos de entradas/archivos.



| | |
|---|---|
| Entrada externa ( Entradas ) | Contar cada datos de usuario o tipo de entrada de control de usuario que (i) entra en el limite extreme del Sistema de software que se esta midiendo y (j) Agrega o cambia los datos en un archive interno logico interno. |
| Salida externa ( salidas ) | Contar cada tipo de salida de datos de usuario único o de control que deja el límite externo del sistema de software que se está midiendo . |
| De archivo logico interno (archivos) | Contar cada grupo lógico principal de datos de usuario o información de control en el sistema de software como un tipo de archivo interno lógico. Incluye cada archivo lógico (por ejemplo , cada grupo lógico de datos) que se genera , utiliza , o mantenido por el sistema de software . |
| Archivos de interface externa (interfaces) | Pasaron archivos o compartidos entre los sistemas de software deben ser contados como los tipos de archivos de interfaz externos dentro de cada sistema. |
| Su mensaje externo (consultas). | Contar cada combinación de entrada-salida única, donde una entrada provoca y genera una salida inmediata , como un tipo de consulta externa |

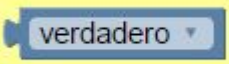
From Cocomo II User Manual via software

## B. Características factor de complejidad.

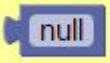
| CARACTERÍSTICA | DESCRIPCIÓN |
|---|---|
| **Comunicación de datos** | Cuántas facilidades de comunicación hay disponibles para ayudar en el intercambio de información con la aplicación o el sistema? |
| **Procesamiento distribuido de datos** | Cómo se manejan los datos y las funciones de procesamiento distribuido |
| **Rendimiento** | Existen requerimientos de velocidad o tiempo de respuesta? |
| **Configuraciones fuertemente utilizadas** | Cómo de intensivas se utilizan las plataformas hardware donde se ejecuta el sistema |
| **Frecuencia de transacciones** | Con qué frecuencia se ejecutan las transacciones? Diariamente, semanalmente,... |
| **Entrada de datos on- line** | Qué porcentaje de la información se ingresa on-line' |
| **Eficiencia del usuario final** | Aplicación diseñada para maximizar la eficiencia del usuario final |
| **Actualizaciones Online** | Cuántos Archivos Lógicos Internos se actualizan por una transacción on-line? |
| **Procesamiento complejo** | Hay procesamientos lógicos o matemáticos intensivos en la aplicación' |
| **Reusabilidad** | La aplicación se desarrolla para suplir una o muchas de las necesidades de los usuarios? |
| **Facilidad de instalación** | Qué tan difícil es la instalación y la conversión al nuevo sistema? |
| **Facilidad de operación** | Cómo de efectivos y/o automatizados deben ser los procedimientos de arranque, parada, backup y restore |
| **Instalación en distintos lugares** | La aplicación fue concebida para su instalación en múltiples sitios y organizaciones? |
| **Facilidad de cambio** | La aplicación fue concebida para facilitar los cambios sobre la misma? |

## C. Definición de bloques.

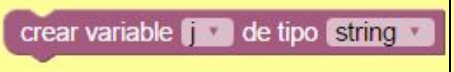
<table>
<tr><th>Id</th><th>Visualización</th><th>Vista</th><th>Modelo</th></tr>
<tr><td colspan="4">Consola</td></tr>
<tr><td>webc_print_num</td><td> 
</td><td>
<table>
<tr><th>Input Id</th><th>Tipo</th></tr>
<tr><td>NUM</td><td>Number</td></tr>
</table>
<table>
<tr><th>Conexión</th><th>Tipo</th></tr>
<tr><td>Anterior</td><td>ALL</td></tr>
<tr><td>Posterior</td><td>ALL</td></tr>
</table>
</td><td>
<table>
<tr><th>Return</th></tr>
<tr><td>-</td></tr>
</table>
<table>
<tr><th colspan="2">Instrucción</th></tr>
<tr><td rowspan="2">Obj</td><td>kind</td></tr>
<tr><td>value</td></tr>
</table>
</td></tr>
<tr><td>webc_print_str</td><td> 
</td><td>
<table>
<tr><th>Input Id</th><th>Tipo</th></tr>
<tr><td>TEXT</td><td>String</td></tr>
</table>
<table>
<tr><th>Conexión</th><th>Tipo</th></tr>
<tr><td>Anterior</td><td>ALL</td></tr>
<tr><td>Posterior</td><td>ALL</td></tr>
</table>
</td><td>
<table>
<tr><th>Return</th></tr>
<tr><td>-</td></tr>
</table>
<table>
<tr><th colspan="2">Instrucción</th></tr>
<tr><td rowspan="3">Obj</td><td>kind</td></tr>
<tr><td>name</td></tr>
<tr><td>value</td></tr>
</table>
</td></tr>
<tr><td colspan="4">Logica</td></tr>
<tr><td>webc_if</td><td> 
</td><td>
<table>
<tr><th>Input Id</th><th>Tipo</th></tr>
<tr><td>NUM</td><td>Boolean</td></tr>
<tr><td>DO</td><td>Statement</td></tr>
</table>
<table>
<tr><th>Conexión</th><th>Tipo</th></tr>
<tr><td>Anterior</td><td>ALL</td></tr>
<tr><td>Posterior</td><td>ALL</td></tr>
</table>
</td><td>
<table>
<tr><th>Return</th></tr>
<tr><td>-</td></tr>
</table>
<table>
<tr><th colspan="2">Instrucción</th></tr>
<tr><td rowspan="3">Obj</td><td>kind</td></tr>
<tr><td>arg1</td></tr>
<tr><td>value</td></tr>
</table>
</td></tr>
</table>

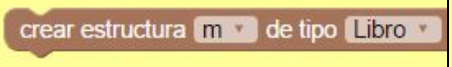
<table>
<tr>
<td>webc_if Else</td>
<td> 
</td>
<td>
<table>
<tr><th>Input Id</th><th>Tipo</th></tr>
<tr><td>BOOL</td><td>Boolean</td></tr>
<tr><td>DO</td><td>Statement</td></tr>
<tr><td>ELSE</td><td>Statement</td></tr>
</table>
<table>
<tr><th>Conexión</th><th>Tipo</th></tr>
<tr><td>Anterior</td><td>ALL</td></tr>
<tr><td>Posterior</td><td>ALL</td></tr>
</table>
</td>
<td>
<table>
<tr><th>Return</th></tr>
<tr><td>-</td></tr>
</table>
<table>
<tr><th colspan="2">Instrucción</th></tr>
<tr><td rowspan="3">Obj</td><td>kind</td></tr>
<tr><td>arg1</td></tr>
<tr><td>value</td></tr>
</table>
</td>
</tr>
<tr>
<td>webc_logic_compare</td>
<td> 
</td>
<td>
<table>
<tr><th>Input Id</th><th>Tipo</th></tr>
<tr><td>A</td><td>Number</td></tr>
<tr><td>B</td><td>Number</td></tr>
</table>
<table>
<tr><th>Conexión</th><th>Tipo</th></tr>
<tr><td colspan="2">-</td></tr>
</table>
</td>
<td>
<table>
<tr><th>Return</th></tr>
<tr><td>Boolean</td></tr>
</table>
<table>
<tr><th colspan="2">Instrucción</th></tr>
<tr><td rowspan="4">Obj</td><td>kind</td></tr>
<tr><td>arg1</td></tr>
<tr><td>arg2</td></tr>
<tr><td>value</td></tr>
</table>
</td>
</tr>
<tr>
<td>webc_logic_operation</td>
<td> 
</td>
<td>
<table>
<tr><th>Input Id</th><th>Tipo</th></tr>
<tr><td>A</td><td>Number</td></tr>
<tr><td>B</td><td>Number</td></tr>
</table>
<table>
<tr><th>Conexión</th><th>Tipo</th></tr>
<tr><td colspan="2">-</td></tr>
</table>
</td>
<td>
<table>
<tr><th>Return</th></tr>
<tr><td>Boolean</td></tr>
</table>
<table>
<tr><th colspan="2">Instrucción</th></tr>
<tr><td rowspan="4">Obj</td><td>kind</td></tr>
<tr><td>arg1</td></tr>
<tr><td>arg2</td></tr>
<tr><td>value</td></tr>
</table>
</td>
</tr>
<tr>
<td>webc_logic_boolean</td>
<td> 
</td>
<td>
<table>
<tr><th>Input Id</th><th>Tipo</th></tr>
</table>
</td>
<td>
<table>
<tr><th>Return</th></tr>
</table>
</td>
</tr>
</table>

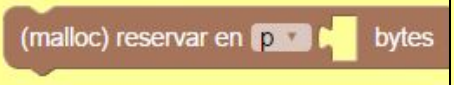
| | | | |
|---|---|---|---|
| | | - | Boolean |
| | | **Conexión** / **Tipo**: - | **Instrucción**: Obj — kind, value |
| **Bucles** | | | |
| webc_while |   | **Input Id** / **Tipo**: BOOL — Boolean<br>**Conexión** / **Tipo**: Anterior — ALL; Posterior — ALL | **Return**: -<br>**Instrucción**: Obj — kind, arg1, value |
| webc_for |   | **Input Id** / **Tipo**: FROM — Number; TO — Number<br>**Conexión** / **Tipo**: Anterior — ALL; Posterior — ALL | **Return**: -<br>**Instrucción**: Obj — kind, name, arg1, value |
| **Datos** | | | |
| webc_math_number |   | **Input Id** / **Tipo**: - | **Return**: Number<br>**Conexión** / **Tipo** / **Instrucción** |

<table>
<tr><td></td><td></td><td><table><tr><td>-</td></tr></table></td><td><table><tr><td rowspan="2">Obj</td><td>kind</td></tr><tr><td>value</td></tr></table></td></tr>
<tr><td>webc_math_arithmetic</td><td>  </td><td><table><tr><th>Input Id</th><th>Tipo</th></tr><tr><td>A</td><td>Number</td></tr><tr><td>B</td><td>Number</td></tr></table><table><tr><th>Conexión</th><th>Tipo</th></tr><tr><td colspan="2">-</td></tr></table></td><td><table><tr><th>Return</th></tr><tr><td>Number</td></tr></table><table><tr><th colspan="2">Instrucción</th></tr><tr><td rowspan="4">Obj</td><td>kind</td></tr><tr><td>arg1</td></tr><tr><td>arg2</td></tr><tr><td>value</td></tr></table></td></tr>
<tr><td>webc_null</td><td>  </td><td><table><tr><th>Input Id</th><th>Tipo</th></tr><tr><td colspan="2">-</td></tr></table><table><tr><th>Conexión</th><th>Tipo</th></tr><tr><td colspan="2">-</td></tr></table></td><td><table><tr><th>Return</th></tr><tr><td>Number</td></tr></table><table><tr><th colspan="2">Instrucción</th></tr><tr><td rowspan="2">Obj</td><td>kind</td></tr><tr><td>value</td></tr></table></td></tr>
<tr><td colspan="4">Arrays -> [ ]</td></tr>
<tr><td>webc_array_one_resize_</td><td>  </td><td><table><tr><th>Input Id</th><th>Tipo</th></tr><tr><td>INDEX</td><td>Number</td></tr></table><table><tr><th>Conexión</th><th>Tipo</th></tr><tr><td>Anterior</td><td>ALL</td></tr><tr><td>Posterior</td><td>ALL</td></tr></table></td><td><table><tr><th>Return</th></tr><tr><td>-</td></tr></table><table><tr><th colspan="2">Instrucción</th></tr><tr><td rowspan="4">Obj</td><td>kind</td></tr><tr><td>arg1</td></tr><tr><td>options</td></tr><tr><td>value</td></tr></table></td></tr>
</table>

<table>
<tr>
<td>webc_array_one_get</td>
<td> 
</td>
<td>
<table>
<tr><th>Input Id</th><th>Tipo</th></tr>
<tr><td>INDEX</td><td>Number</td></tr>
</table>
<table>
<tr><th>Conexión</th><th>Tipo</th></tr>
<tr><td>Anterior</td><td>ALL</td></tr>
<tr><td>Posterior</td><td>ALL</td></tr>
</table>
</td>
<td>
<table>
<tr><th>Return</th></tr>
<tr><td>Number</td></tr>
</table>
<table>
<tr><th colspan="2">Instrucción</th></tr>
<tr><td rowspan="3">Obj</td><td>kind</td></tr>
<tr><td>arg1</td></tr>
<tr><td>value</td></tr>
</table>
</td>
</tr>
<tr>
<td>webc_array_one_set</td>
<td> 
</td>
<td>
<table>
<tr><th>Input Id</th><th>Tipo</th></tr>
<tr><td>INDEX</td><td>Number</td></tr>
<tr><td>VALUE</td><td>Number</td></tr>
</table>
<table>
<tr><th>Conexión</th><th>Tipo</th></tr>
<tr><td>Anterior</td><td>ALL</td></tr>
<tr><td>Posterior</td><td>ALL</td></tr>
</table>
</td>
<td>
<table>
<tr><th>Return</th></tr>
<tr><td>-</td></tr>
</table>
<table>
<tr><th colspan="2">Instrucción</th></tr>
<tr><td rowspan="4">Obj</td><td>kind</td></tr>
<tr><td>arg1</td></tr>
<tr><td>name</td></tr>
<tr><td>value</td></tr>
</table>
</td>
</tr>
<tr><td colspan="4">Arrays -> [ ] [ ]</td></tr>
<tr>
<td>webc_array_two_resize</td>
<td> 
</td>
<td>
<table>
<tr><th>Input Id</th><th>Tipo</th></tr>
<tr><td>INDEX1</td><td>Number</td></tr>
<tr><td>INDEX2</td><td>Number</td></tr>
</table>
<table>
<tr><th>Conexión</th><th>Tipo</th></tr>
<tr><td>Anterior</td><td>ALL</td></tr>
<tr><td>Posterior</td><td>ALL</td></tr>
</table>
</td>
<td>
<table>
<tr><th>Return</th></tr>
<tr><td>-</td></tr>
</table>
<table>
<tr><th colspan="2">Instrucción</th></tr>
<tr><td rowspan="5">Obj</td><td>kind</td></tr>
<tr><td>arg1</td></tr>
<tr><td>arg2</td></tr>
<tr><td>name</td></tr>
<tr><td>value</td></tr>
</table>
</td>
</tr>
<tr>
<td>webc_array_two_get</td>
<td> 
</td>
<td>
<table>
<tr><th>Input Id</th><th>Tipo</th></tr>
</table>
</td>
<td>
<table>
<tr><th>Return</th></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td></td>
<td>
<table>
<tr><td>INDEX1</td><td>Number</td></tr>
<tr><td>INDEX2</td><td>Number</td></tr>
</table>
<table>
<tr><th>Conexión</th><th>Tipo</th></tr>
<tr><td colspan="2">-</td></tr>
</table>
</td>
<td>
<table>
<tr><td>-</td></tr>
</table>
<table>
<tr><th colspan="2">Instrucción</th></tr>
<tr><td rowspan="4">Obj</td><td>kind</td></tr>
<tr><td>arg1</td></tr>
<tr><td>arg2</td></tr>
<tr><td>name</td></tr>
</table>
</td>
</tr>
<tr>
<td>webc_array_two_set</td>
<td> 
</td>
<td>
<table>
<tr><th>Input Id</th><th>Tipo</th></tr>
<tr><td>INDEX1</td><td>Number</td></tr>
<tr><td>INDEX2</td><td>Number</td></tr>
<tr><td>VALUE</td><td>Number</td></tr>
</table>
<table>
<tr><th>Conexión</th><th>Tipo</th></tr>
<tr><td>Anterior</td><td>ALL</td></tr>
<tr><td>Posterior</td><td>ALL</td></tr>
</table>
</td>
<td>
<table>
<tr><th>Return</th></tr>
<tr><td>-</td></tr>
</table>
<table>
<tr><th colspan="2">Instrucción</th></tr>
<tr><td rowspan="5">Obj</td><td>kind</td></tr>
<tr><td>name</td></tr>
<tr><td>arg1</td></tr>
<tr><td>arg2</td></tr>
<tr><td>value</td></tr>
</table>
</td>
</tr>
<tr><th colspan="4">Variables</th></tr>
<tr>
<td>webc_variables_set</td>
<td> 
</td>
<td>
<table>
<tr><th>Input Id</th><th>Tipo</th></tr>
<tr><td colspan="2">-</td></tr>
</table>
<table>
<tr><th>Conexión</th><th>Tipo</th></tr>
<tr><td>Anterior</td><td>ALL</td></tr>
<tr><td>Posterior</td><td>ALL</td></tr>
</table>
</td>
<td>
<table>
<tr><th>Return</th></tr>
<tr><td>-</td></tr>
</table>
<table>
<tr><th colspan="2">Instrucción</th></tr>
<tr><td rowspan="5">Obj</td><td>kind</td></tr>
<tr><td>name</td></tr>
<tr><td>type</td></tr>
<tr><td>value</td></tr>
</table>
</td>
</tr>
<tr>
<td>webc_variables_get</td>
<td> 
</td>
<td>
<table>
<tr><th>Input Id</th><th>Tipo</th></tr>
<tr><td colspan="2">-</td></tr>
</table>
</td>
<td>
<table>
<tr><th>Return</th></tr>
<tr><td>Number</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr><td></td><td></td><td><table><tr><th>Conexión</th><th>Tipo</th></tr><tr><td colspan="2">-</td></tr></table></td><td><table><tr><th colspan="2">Instrucción</th></tr><tr><td rowspan="2">Obj</td><td>kind</td></tr><tr><td>name</td></tr></table></td></tr>
<tr><td colspan="4">Estructuras -> Definición</td></tr>
<tr><td>webc_struct_set</td><td> 
</td><td><table><tr><th>Input Id</th><th>Tipo</th></tr><tr><td>NAME</td><td>String</td></tr><tr><td>STRUCT</td><td>Statement</td></tr></table><table><tr><th>Conexión</th><th>Tipo</th></tr><tr><td colspan="2">-</td></tr></table></td><td><table><tr><th>Return</th></tr><tr><td>-</td></tr></table><table><tr><th colspan="2">Instrucción</th></tr><tr><td rowspan="2">Obj</td><td>kind</td></tr><tr><td>value</td></tr></table></td></tr>
<tr><td>webc_struct_addField</td><td> 
</td><td><table><tr><th>Input Id</th><th>Tipo</th></tr><tr><td colspan="2">-</td></tr></table><table><tr><th>Conexión</th><th>Tipo</th></tr><tr><td>Anterior</td><td>STRUCTFIELD</td></tr><tr><td>Posterior</td><td>STRUCTFIELD</td></tr></table></td><td><table><tr><th>Return</th></tr><tr><td>-</td></tr></table><table><tr><th colspan="2">Instrucción</th></tr><tr><td rowspan="3">Obj</td><td>kind</td></tr><tr><td>name</td></tr><tr><td>type</td></tr></table></td></tr>
<tr><td colspan="4">Estructuras -> Instanciación</td></tr>
<tr><td>webc_struct_create</td><td> 
</td><td><table><tr><th>Input Id</th><th>Tipo</th></tr><tr><td colspan="2">-</td></tr></table><table><tr><th>Conexión</th><th>Tipo</th></tr><tr><td>Anterior</td><td>ALL</td></tr><tr><td>Posterior</td><td>ALL</td></tr></table></td><td><table><tr><th>Return</th></tr><tr><td>-</td></tr></table><table><tr><th colspan="2">Instrucción</th></tr><tr><td rowspan="2">Obj</td><td>kind</td></tr><tr><td>name</td></tr></table></td></tr>
</table>

<table>
<tr><td></td><td></td><td></td><td><table><tr><td></td><td>type</td></tr><tr><td></td><td>value</td></tr></table></td></tr>
<tr><td colspan="4">Punteros</td></tr>
<tr><td>webc_pointers_set</td><td> 
</td><td><table><tr><th>Input Id</th><th>Tipo</th></tr><tr><td colspan="2">-</td></tr></table><table><tr><th>Conexión</th><th>Tipo</th></tr><tr><td>Anterior</td><td>ALL</td></tr><tr><td>Posterior</td><td>ALL</td></tr></table></td><td><table><tr><th>Return</th></tr><tr><td>-</td></tr></table><table><tr><th colspan="2">Instrucción</th></tr><tr><td rowspan="4">Obj</td><td>kind</td></tr><tr><td>name</td></tr><tr><td>type</td></tr><tr><td>value</td></tr></table></td></tr>
<tr><td>webc_pointer_change</td><td> 
</td><td><table><tr><th>Input Id</th><th>Tipo</th></tr><tr><td colspan="2">-</td></tr></table><table><tr><th>Conexión</th><th>Tipo</th></tr><tr><td>Anterior</td><td>ALL</td></tr><tr><td>Posterior</td><td>ALL</td></tr></table></td><td><table><tr><th>Return</th></tr><tr><td>-</td></tr></table><table><tr><th colspan="2">Instrucción</th></tr><tr><td rowspan="4">Obj</td><td>kind</td></tr><tr><td>pointer Or</td></tr><tr><td>pointer Dest</td></tr><tr><td>value</td></tr></table></td></tr>
<tr><td>webc_malloc</td><td> 
</td><td><table><tr><th>Input Id</th><th>Tipo</th></tr><tr><td>VAR</td><td>Number</td></tr></table><table><tr><th>Conexión</th><th>Tipo</th></tr><tr><td>Anterior</td><td>ALL</td></tr><tr><td>Posterior</td><td>ALL</td></tr></table></td><td><table><tr><th>Return</th></tr><tr><td>-</td></tr></table><table><tr><th colspan="2">Instrucción</th></tr><tr><td rowspan="2">Obj</td><td>kind</td></tr><tr><td>pointer Or</td></tr></table></td></tr>
</table>

<table>
<tr><td></td><td></td><td></td><td><table><tr><td></td><td>value</td></tr></table></td></tr>
<tr><td>webc_free</td><td> 
</td><td><table><tr><th>Input Id</th><th>Tipo</th></tr><tr><td colspan="2">-</td></tr></table><br><table><tr><th>Conexión</th><th>Tipo</th></tr><tr><td>Anterior</td><td>ALL</td></tr><tr><td>Posterior</td><td>ALL</td></tr></table></td><td><table><tr><th>Return</th></tr><tr><td>-</td></tr></table><br><table><tr><th colspan="2">Instrucción</th></tr><tr><td rowspan="3">Obj</td><td>kind</td></tr><tr><td>pointer</td></tr><tr><td>value</td></tr></table></td></tr>
</table>